

JOHN CAMPBELL

INSIDE  
APPLE'S<sup>®</sup>  
ProDOS<sup>™</sup>





INSIDE  
APPLE'S®  
PRODOS™

# INSIDE APPLE'S® PRODOS™.

*John Campbell*



A Reston Computer Group Book  
Reston Publishing, Inc.  
*A Prentice-Hall Company*  
Reston, Virginia



Campbell, J. L. (John L.)  
Inside Apple's ProDOS.

"A Reston Computer Group book."

Bibliography: p.

Includes index.

1. ProDOS (Computer operating system) 2. Apple II  
(Computer)--Programming. 3. Basic (Computer program  
language) I. Title. II. Title: Inside Apple's ProD.O.S.  
QA766.C343 1984 001.64'2 84-6988  
ISBN 0-8359-3078-5

The terms Apple, Apple II, Apple II Plus, Apple IIe, Apple III, Disk II, DOS 3.3, ProFile, ProDOS, Integer BASIC, Applesoft II BASIC, The Filer, CONVERT, EXERCISER, and The APA are either registered trademarks, trademarks, or copyrighted by Apple Computer, Inc.

Interior design and production by Carolyn Ormes

© 1984 by John L. Campbell

All rights reserved. No part of this book may be reproduced in any way or by any means, without permission in writing from the author.

10 9 8 7 6 5 4 3 2 1

PRINTED IN THE UNITED STATES OF AMERICA

# CONTENTS

*Nothing is more expensive*

*than a start.*

*Nietzsche, 1888*

Preface xi

## **CHAPTER 1: INTRODUCTION 1**

1.0. Overview 1

1.1. Introducing ProDOS 3

1.1.1. What is Needed 5

1.1.2. The ProDOS Diskettes 6

1.1.3. Making a Startup Diskette 8

1.1.4. The ProDOS Programs 00

1.2. Make a ProDOS Copy 13

1.3. ProDOS Commands Described 16

1.4. Starting ProDOS 18

1.4.1. From a Cold Start 19

1.4.2. From Other Ways 20

- 1.5. The HELP Command 21
- 1.6. ProDOS and DOS 22
- Summary 23
- Questions 24

## **CHAPTER 2: PRODOS FILES AND COMMANDS 25**

- 2.0. Overview 25
- 2.1. How ProDOS Works 26
  - 2.1.1. A Startup Diskette 27
  - 2.1.2. The PRODOS Program 28
  - 2.1.3. The BASIC SYSTEM Program 29
  - 2.1.4. The STARTUP Program 29
- 2.2. Volumes and Files 30
  - 2.2.1. The Directory 31
  - 2.2.2. The Pathname 36
  - 2.2.3. The Prefix 38
- 2.3. ProDOS and Programs 39
  - 2.3.1. The—(DASH) Command 39
  - 2.3.2. The RUN Command 40
  - 2.3.3. The LOAD Command 42
  - 2.3.4. The SAVE Command 43
- Summary 43
- Questions 45

## **CHAPTER 3: HOUSEKEEPING COMMANDS 46**

- 3.0. Overview 46
- 3.1. The CAT and CATALOG Commands 47
- 3.2. The PREFIX Command 49
- 3.3. The CREATE Command 52
- 3.4. The RENAME Command 54
- 3.5. The DELETE Command 55
- 3.6. The LOCK and UNLOCK Command 55
- 3.7. I/O from Programs 57
  - 3.7.1. The CHAIN Command 57
  - 3.7.2. The STORE Command 57
  - 3.7.3. The RESTORE Command 58
  - 3.7.4. The PR# Command 59
  - 3.7.5. The IN# Command 60
- Summary 61
- Questions 62

**CHAPTER 4: SEQUENTIAL-ACCESS FILES 63**

- 4.0. Overview 63
- 4.1. Sequential-access Files 65
- 4.2. The OPEN Command 72
- 4.3. The READ Command 74
- 4.4. The WRITE Command 75
- 4.5. The CLOSE Command 76
- 4.6. The APPEND Command 76
- 4.7. The FLUSH Command 77
- 4.8. The POSITION Command 78
- Summary 78
- Questions 79

**CHAPTER 5: RANDOM ACCESS FILES 80**

- 5.0. Overview 80
- 5.1. Random-Access Files 81
  - 5.1.1. The Record Length 82
  - 5.1.2. Writing a Record 82
  - 5.1.3. Record Character Storage 83
  - 5.1.4. Reading from a Record 89
- 5.2. The OPEN Command 89
- 5.3. The READ Command 90
- 5.4. The WRITE Command 91
- 5.5. The CLOSE Command 92
- 5.6. The DELETE Command 93
- 5.7. The APPEND Command 93
- 5.8. The FLUSH Command 94
- Summary 95
- Questions 95

**CHAPTER 6: BINARY FILES 96**

- 6.0. Overview 96
- 6.1. Binary Files 97
  - 6.1.1. Binary Addresses 98
  - 6.1.2. Command Options 98
- 6.2. The BLOAD Command 99
  - 6.2.1. Installing Machine-code Routines 100
- 6.3. The BSAVE Command 101
- 6.4. The BRUN Command 102
- 6.5. The PR# and IN# Command 103
- 6.6. The Monitor and ProDOS 104
- Summary 105
- Questions 106

## **CHAPTER 7: EXECUTIVE FILES 107**

- 7.0. Overview 107
- 7.1. EXEC Files Demonstration 108
- 7.2. The EXEC Command 110
- 7.3. EXEC Uses 111
- Summary 111
- Questions 111

## **CHAPTER 8: THE PRODOS FILER AND CONVERT PROGRAMS 112**

- 8.0. Overview 112
- 8.1. Using the FILER Program 115
  - 8.1.1. ProDOS FILER Menu 115
  - 8.1.2. TUTOR 116
  - 8.1.3. File Commands 117
    - LIST PRODOS DIRECTORY 117
    - COPY FILES 119
    - DELETE FILES 120
    - COMPARE FILES 121
    - ALTER WRITE-PROTECTION 121
    - RENAME FILES 121
    - MAKE DIRECTORY 121
    - SET PREFIX 122
  - 8.1.4. Volume Commands 123
    - FORMAT A VOLUME 125
    - COPY A VOLUME 127
    - LIST VOLUMES 128
    - RENAME A VOLUME 129
    - DETECT BAD BLOCKS 130
    - BLOCK ALLOCATION 132
    - COMPARE VOLUMES 133
  - 8.1.5. Configuration Defaults 134
    - SELECT DEFAULTS 134
    - RESTORE DEFAULTS 135
  - 8.1.6. QUIT 135
- 8.2. Using the CONVERT Program 136
  - 8.2.1. ProDOS CONVERT Menu 136
  - 8.2.2. Reverse Transfer Direction 137
  - 8.2.3. Change DOS 3.3 Slot and Drive 137
  - 8.2.4. Set ProDOS Prefix 139
  - 8.2.5. Set ProDOS Date 139
  - 8.2.6. Transfer or List Files 139

**8.3. FILER and CONVERT Error Messages 139**

Summary 139

Questions 140

**CHAPTER 9: THE MACHINE LANGUAGE INTERFACE 141****9.0. Overview 141****9.1. Memory Usage 142**

## 9.1.1. ProDOS Loading Sequence 142

## 9.1.2. Memory Maps 143

**9.2. Issuing Calls 152**

## 9.2.1. Housekeeping Calls 154

## 9.2.2. Filing Calls 163

## 9.2.3. System Calls 173

## 9.2.4. Direct Access Calls 174

**9.3. MLI Error Codes 176****9.4. Writing a System Program 177**

Summary 178

Questions 178

**CHAPTER 10: BASIC PROGRAMMING SYSTEMS 180****10.0. Overview 180****10.1. Designing the Program 181****10.2. Program Modules 185**

## 10.2.1. The Skeleton Program 185

## 10.2.2. Number and String Module 191

## 10.2.3. Pathname Determination Subroutine 194

## 10.2.4. Print to Screen Subroutine 196

## 10.2.5. Read a Record Subroutines 196

## 10.2.6. Write a Record Subroutines 197

## 10.2.7. System Configuration Setup Subroutines 198

## 10.2.8. Thunderclock Routine 199

## 10.2.9. Read Current Prefix Routine 200

## 10.2.10. Report Heading Subroutine 200

**10.3. The Program 201**

Summary 227

Questions 228

**APPENDICES 229****A. DOS 3.3 and ProDOS 1.0 Comparisons 229**

Overview 229

The Diskettes 230

Commands no Longer Supported 231

ProDOS Commands Supported 231

<b>B. ProDOS 1.0 Memory Map</b>	<b>241</b>
Overview	241
High Memory Considerations	242
Zero Page	242
General Memory Map	248
<b>C. ProDOS Command Summary</b>	<b>250</b>
<b>D. Error Messages</b>	<b>256</b>
Overview	256
Determining the ERROR	257
ProDOS Messages	257
FILER and CONVERT Messages	259
Applesoft II Messages	264
<b>E. Miscellaneous Topics</b>	<b>266</b>
Overview	266
Clock/Calendar Card	266
/RAM in the Apple IIe	268
ProFile Hard Disk Storage	270
<b>F. ProDOS, Apple III, and SOS</b>	<b>271</b>
Overview	271
ProDOS and SOS	271
File Comparisons	272
System Comparisons	272
<b>G. The APA Program</b>	<b>274</b>
Overview	274
Starting APA	274
AUTO Command	275
MANUAL Command	276
RENUMBER Command	276
HOLD Command	277
MERGE Command	277
COMPRESS Command	278
SHOW Command	278
NOSHOW Command	279
LENGTH Command	279
XREF Command	279
CONVERT Command	280
EXIT Command	280
<b>H. ASCII Character Codes</b>	<b>282</b>
<b>I. Hexadecimal to Decimal Conversions</b>	<b>286</b>

## **BIBLIOGRAPHY 288**

## **GLOSSARY 292**

## **INDEX 299**

# PREFACE

This book describes the new Professional Disk Operating System (ProDOS) recently introduced by Apple Computer, Inc. This new operating system is more than just another new computer disk operating program like the changes made when DOS 3.3 was introduced. ProDOS is a completely new computer system operating environment. ProDOS contains new commands, expanded and improved old commands, file management utilities, assembler, data types, file types, and new procedures. It's a new, exciting, environment that adds materially to the capability of the Apple II® family of computers.

A disk operating system is a computer program or set of programs that serve as the executive manager for the information stored on diskettes. The operating system allows you to store, retrieve, or rearrange information on diskettes. ProDOS allows you to organize your information stored on all of your Apple II family of diskettes. Throughout, comparisons will be made to DOS 3.3 so that you will be able to make the transition to ProDOS easily.

As you begin reading this book, you will first be introduced to the ProDOS operating system parts in Chapter 1. This chapter outlines the conventions used throughout the book for describing ProDOS.



To use ProDOS you will need an Apple II family computer with a least 64K of memory, Applesoft II BASIC® in Read Only Memory (ROM), and at least one Disk II disk drive. More will be said about these requirements in Chapter 1.

Chapter 2 discusses the files, file types, command syntax, and your first, most often used ProDOS commands.

The general system housekeeping commands are discussed in Chapter 3.

Chapters 4 through 7 discuss each of the major types of files you will be using. These include:

- Sequential access files
- Random access files
- Binary files
- Executive files

Chapter 8 will discuss the Filer program. This program performs a number of different volume and file utility functions.

Chapter 9 briefly discusses the machine language interface portion of ProDOS. This allows you to customize your systems operation.

The last chapter will put together a number of routines that you can use in your own programs. These are used in a program as the vehicle of presentation.

With this new operating system, all of the Apple computer system operating environments operate in a similar manner. I wish to thank Apple Computer, Inc. for permission to quote directly from their many useful manuals.

I hope that you will enjoy this book.

J. L. Campbell

**INSIDE  
APPLE'S®  
PRODOS™.**

# I. INTRODUCTION

*If a system injures the intelligence  
it is bad. If it injures the character  
it is vicious, if it injures the  
conscience it is criminal.*

*Henri Frederic Amiel, 1852*

## 1.0. OVERVIEW

This chapter introduces you to Apple's ProDOS™. (Professional Disk Operating System).

After a brief discussion of some of the conventions used throughout this book, you will be introduced to the interrelationship of various software programs and how they operate to perform all of those things that allow you to save your balanced checkbook transactions correctly. You will be given a brief description of their capabilities and functions.

Of course, you should know early on just what you need in the way of a computer system to take advantage of ProDOS. The physical system configuration requirements for each of the Apple® II family of computers will be outlined. By "Apple II family," I mean the Apple II, Apple® II Plus, and Apple® IIe computers.

By the time you have read to Section 1.2, you should have enough information to be able to begin exercising your system with the new operating system. The diskettes that came with the operating system will be discussed. "Diskette" refers to a 5 1/4-inch floppy disk. Then, you will create a working diskette of your own so that, when working with the programs and pro-

gram segments in this book, you may code them, save them, and try them out to see what happens.

You will make backup copies of the original diskettes using the **FILER** program and explore some of the programs on the original diskettes. The term "backup" means a duplicate. You will be told how to power up your system and get all of the pieces operating together.

You will explore one of the more interesting new capabilities of ProDOS, called **HELP**. **HELP** is a program that uses the text file **HELPSCREENS** stored on one of your original diskettes. The **HELP** program is discussed in Section 1.5. It is comforting to know that there is help when you forget just what is required.

Finally, the differences between DOS 3.3 and ProDOS will be discussed for the first time. Throughout this book, the similarities and differences between these two operating systems will be shown in each instance where they are applicable.

Every time the Apple II, Apple II Plus, and Apple IIe computers are discussed, the term "Apple computers" will be used. ProDOS applies to all of these computers equally, provided each individual machine is configured correctly to handle ProDOS. These individual configurations will be outlined in Section 1.1.

Another convention used throughout is to draw the distinction between this new operating system (ProDOS) and the operating system program (PRODOS). At this point you need not know or understand the differences between these two terms. Their differences will become clear very shortly.

When discussing a computer application or operating system program, the name of that program will be capitalized. Further, operating system commands will be shown capitalized.

There are two Apple computer languages mentioned in this book: Integer BASIC and Applesoft II BASIC for the Apple computers. These are Apple's dialects of the BASIC language. The primary emphasis will be on Applesoft II BASIC, since Integer BASIC is *not* supported by ProDOS.

Finally, comparisons will be made between ProDOS and DOS 3.3 pointing out the similarities and differences. This should enable you to make the transition from DOS 3.3 to ProDOS easily and quickly.

There are a number of assumptions that will be made in presenting information in this book. These are:

Your computer is connected up correctly (see Section 1.1.1).

You have a fundamental working knowledge of:

Applesoft ® II BASIC and  
DOS 3.3.

I realize that this may be more than you can handle at this time. With that in mind, I will try to explain things carefully as you proceed through this book. Further, if you feel you need more in-depth knowledge concerning these subjects, it is recommended that you acquire and use my other books, *Programming the Apple: A Structured Approach*, and *Programming Tips and Techniques for the Apple II Plus/Apple IIe*. Besides, I need the money! So buy them, please! Both books are published by the R. J. Brady Co.

In each chapter, where commands are introduced, there is a box that gives the command syntax, provides command examples, and explains in which operating mode each may be used. Syntax means the rules governing the structure of statements, instructions, and commands.

If a command allows the use of options, the options will be shown enclosed in square brackets. Those parts of a command that are required to be used are shown without the square brackets.

In the particular chapter section that describes a command, the specific form, or syntax, will be given for that command. In addition, Appendix C gives a summary of all commands and their options. Each ProDOS command has a number of options, or additional specifications, that may be either defined or not used.

The commands introduced in this chapter are shown in the box below.

HELP	e.g.,	HELP	Immediate mode only
		HELP PREFIX	
NOHELP	e.g.,	NOHELP	Immediate mode only

## 1.1. INTRODUCING PRODOS

Before getting into this new operating system, it is necessary to understand the set of computer programs involved and where they fit in the overall scheme of things. Let's look at a computer system from the point of view of the computer programs operating during the time you are making entries into a general ledger, writing a letter, or fighting a rousing space war marathon. This is actually a logical computer as shown diagrammatically in Figure 1.1.

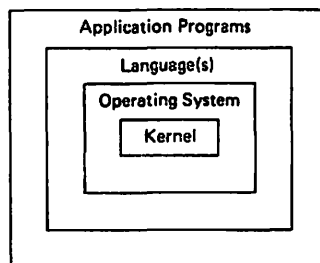


Figure 1.1. Logical computer.

The kernel for the Apple computers actually contains both the MONITOR and AUTO-START programs. The kernel programs usually perform very basic operating functions. The AUTOSTART program allows the Apple computers to “come up running” when power is first applied to your system. A secondary, but equally important, function included in AUTOSTART is the screen editing capability of your Apple computer. There are other general kernel functions performed by the MONITOR set of program modules.

These are:

**Character I/O**

- Get character from keyboard
- Get an input line of characters
- Echo character to the screen
- Generate a carriage return character

**Low-resolution graphics plotting**

- Plot point
- Plot line

**Speaker routines**

- Sound bell
- Beep speaker

**Paddle, button, and annunciator I/O**

- Sense paddle position
- Sense button pushed
- Sense annunciator used

**Cassette tape I/O**

- Load program from tape
- Save program to tape

**Screen commands**

- Clear screen
- Clear screen section
- Set text mode
- Clear line
- Set scrolling window
- Set graphics mode
- Set screen modes
- Normal
- Inverse
- Flashing
- Graphics

**Reset machine**

- 6502 registers
  - Save registers
  - Move memory
  - Restore registers
  - Print register contents

As you can see, there are a number of very important but necessary functions that are performed by the kernel set of program modules and routines.

The next layer or ring of programs normally resident in the computer's memory is the operating system. In this case it will be ProDOS, which is the subject of this book.

The third layer of software activated is the computer language(s) needed by the last or fourth layer. This is Applesoft II BASIC for Apple computers. Although Applesoft II BASIC is not the only language available for Apple computers, it is the only one of interest here.

The outer layer of computer software operating is the application program. An application program is any program you are currently using. It could be a checkbook balancer, general ledger, word processor, or graphics Klingon space war.

In order to accomplish anything with a computer system, many different programs operating on different levels must be active simultaneously. Of all of these programs, the operating system is the only one discussed in detail in this book.

### **1.1.1. What is needed**

In order for you to install ProDOS on your computer system you will need to configure your particular system as follows:

**Apple IIe:**

Computer.

Video display (this may be a TV or video monitor).

Disk II disk drive.

Applesoft II BASIC in ROM (ROM means read only memory).

ProDOS diskettes.

**Apple II Plus:**

Computer with 64K RAM (K stands for 1024 bytes; RAM stands for random access memory).

48K plus 16K language card or

48K plus 16K RAM card.

Video display.

Disk II disk drive.

Applesoft II BASIC in ROM.  
ProDOS diskettes.

Apple II:

Computer with 64K.  
48K plus 16K language card or  
48K plus 16K RAM card.

Video display.

Disk II disk drive.

Applesoft II BASIC in ROM installed on a firmware card (firmware means a program implemented in ROM).

ProDOS diskettes.

If you have one of the early Apple II computers with the Applesoft II BASIC firmware card installed in slot zero, it is a simple matter to exchange the Integer BASIC and MONITOR chips on the motherboard with the Applesoft II BASIC, AUTOSTART, and MONITOR chips on the firmware card. The motherboard is the main circuit board of your computer, which contains the expansion slots. Once that is done, then you only need to have a 16K memory expansion board or language board installed into the vacated slot zero. You can now use ProDOS. Don't throw the Integer BASIC firmware card away. You may want to use it again sometime with software that uses that language and DOS 3.3.

### 1.1.2. The ProDOS diskettes

This section will discuss the two diskettes that come with ProDOS. These diskettes are:

- The /PRODOS diskette
- The /EXAMPLES diskette

The /PRODOS diskette takes the place of the SYSTEM MASTER diskette that came with DOS 3.3. The contents of the diskette are as shown in Figure 1.2, below.

Place the ProDOS diskette into disk drive 1 of your system. Then power up your system. If you do not know how to power up your system, then skip to Section 1.4 for the general power up procedure.

The contents of a diskette may be seen by using the command:

```
]CAT,S6,D1
```

Try this command now that your system is powered up.



## /PRODOS

NAME	TYPE	BLOCKS	MODIFIED
*PRODOS	SYS	29	1-SEP-83
*BASIC.SYSTEM	SYS	21	1-SEP-83
*STARTUP	BAS	7	1-AUG-83
*CONVERT	SYS	42	1-SEP-83
*FILER	SYS	51	1-SEP-83
EXERCISER	SYS	16	29-JUL-83
FORMATTER	TXT	15	4-AUG-83
EDASM.ASM	BIN	29	14-JUN-83
EDASM.ED	BIN	17	14-JUN-83
EDASM.SYSTEM	BIN	9	1-AUG-83
BUGBYTER	BIN	15	4-APR-83
*BASIC.RUNTIME	SYS	21	1-SEP-83

BLOCKS FREE: 1      BLOCKS USED: 279

Figure 1.2. /PRODOS diskette contents.

The actual contents of your /PRODOS diskette may be slightly different because the final production diskette contents will probably change between now and the time ProDOS is shipped to the general buying public.

The most obvious thing to notice that ProDOS will present much more information to you than what is presented with DOS 3.3. Each of these columns of information will be explained in detail later.

Now, let's look at the contents of the /EXAMPLES diskette. This is shown in Figure 1.3. You do this by replacing the diskette in drive 1 with the other diskette that came with ProDOS.

]CAT,S6,D1

## /EXAMPLES

NAME	TYPE	BLOCKS	MODIFIED
*PRODOS	SYS	29	1-SEP-83
*BASIC.SYSTEM	SYS	21	1-SEP-83
STARTUP	BAS	7	22-JUL-83
HELP	BAS	1	1-SEP-83
HELPSCREENS	TXT	60	1-SEP-83
DIRECTORY	DIR	1	28-MAR-83
PRACTICE	DIR	1	18-JUL-83

PROGRAMS	DIR	3	5-OCT-83
DATA	DIR	1	29-AUG-83
EXTRAS	DIR	1	6-OCT-83

BLOCKS FREE: 31      BLOCKS USED: 249

Figure 1.3. /EXAMPLES diskette contents.

Once again, notice the amount of information presented to you. This information may also change slightly by the time you receive your ProDOS system. For now, the only thing to know is what has been presented. Later in this book, each column will be explained in detail.

Now that you have your system running, you are ready to make a ProDOS diskette that will be used to store the programs written in this book.

Search through your diskettes and find at least three that can be reinitialized and used for ProDOS information.

### 1.1.3. Making a startup diskette

Chapter 8 will discuss the FILER and CONVERT programs in detail. However, this section is going to show you how to use the FILER program to create a diskette for your own use. In DOS 3.3 terms, you will initialize a diskette with the boot program HELLO installed. Boot means to start up automatically. In ProDOS terms, you must first format a diskette using the FILER program and then install programs on that diskette. The following programs are to be installed:

- PRODOS
- BASIC.SYSTEM
- STARTUP

Power up your system as outlined in Section 1.4. If you are totally unfamiliar with how to power up your system, then skip to that section, read it, and then return here.

The first thing to do is to place the /PRODOS diskette in your boot drive and power up your system. When the power up procedure has been completed, you will be in the immediate mode with the Applesoft II BASIC prompt character and cursor in the bottom left portion of the screen.

At this point, let's start to create a program diskette of your own. In the following discussion, you will execute a number of commands not knowing the reasons why. Don't worry, they will be explained in other sections or chapters of this book. The first command to be executed is:

]— FILER

The — (DASH) or minus sign key is the shorthand means available in ProDOS for executing a program regardless of program type. The — (DASH) may be used in place of RUN or

BRUN. This command is known as the intelligent run command. This command is discussed in Section 2.3.1. This command is *not* a part of DOS 3.3. FILER is the name of the program that performs a number of volume and file manipulation and utility functions. This program allows you to perform many general system housekeeping chores. By the way, you will find this new — (DASH) command very handy.

Once this command is executed, your boot disk drive will come on while the FILER program is being loaded into memory. After the program is loaded, it will be automatically executed. The first screen that will be presented will be the main FILER menu screen. This screen is shown in Figure 1.4.

From this screen you will want to select the V (VOLUME COMMANDS) option. This is done by pressing the V key on the keyboard without using the RETURN key. Volume commands allow you to work with an entire diskette as a unit. After selecting the V option, you will be presented with the secondary menu as shown in Figure 1.5.

In this case you will choose the F (FORMAT A VOLUME) option. This corresponds to the INIT command in DOS 3.3.

*You may want to remove the diskette in your boot drive BEFORE making this selection.*

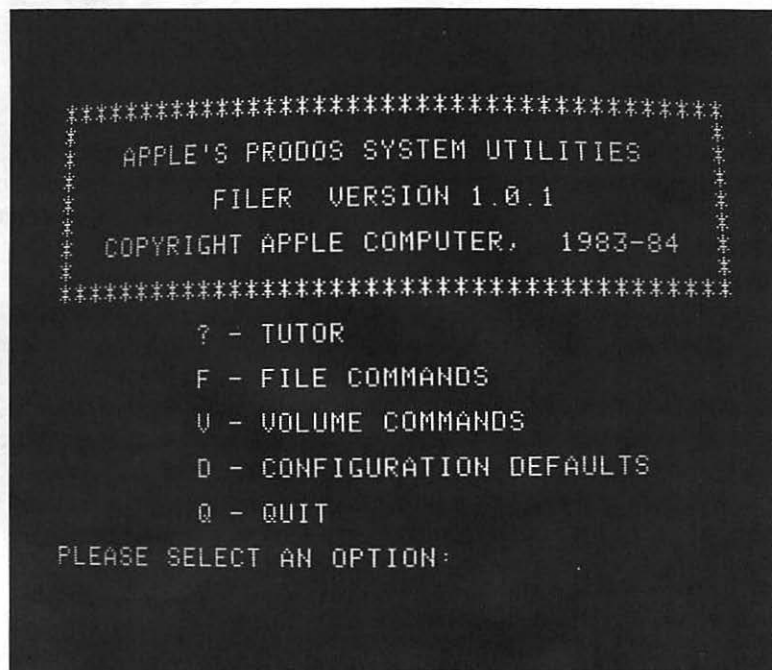


Figure 1.4. FILER main screen.

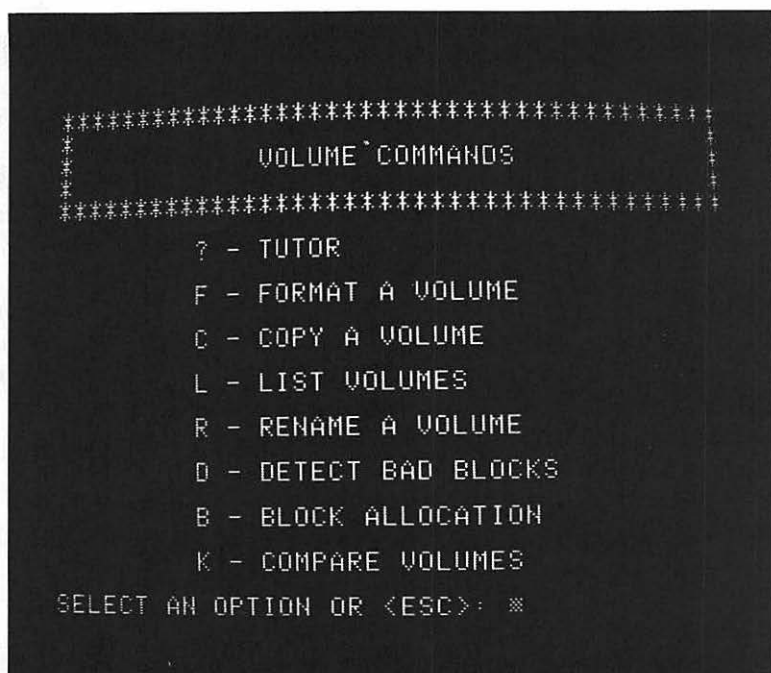


Figure 1.5. Volume commands screen.

By selecting the F option, you will be shown a third screen. This screen, however, is a data entry screen. You will finally be entering the information necessary to format a diskette. This screen is shown in Figure 1.6.

At this point, remove the boot diskette and replace it with a new blank diskette. This new diskette is what you will use throughout this book to store programs and files.

This video screen asks you to enter the slot and drive number for the diskette to be formatted. You have the option of accepting the default values by typing the RETURN key for the values presented, or you may enter your own values. A default value is the value used by the computer program unless another value is provided by the program operator. The defaults are slot 6 and drive 1.

The cursor will then put you in the middle of the screen, where you are to respond by entering the name of the volume (diskette media) you want to format. Once again, you could accept the default of /BLANK00. For now, enter the following:

SCRATCH.DISK

after the slash mark. Notice that there is a slash mark (/) that precedes the default name for the volume. All directory and volume names have the slash mark (/) before the title. This slash

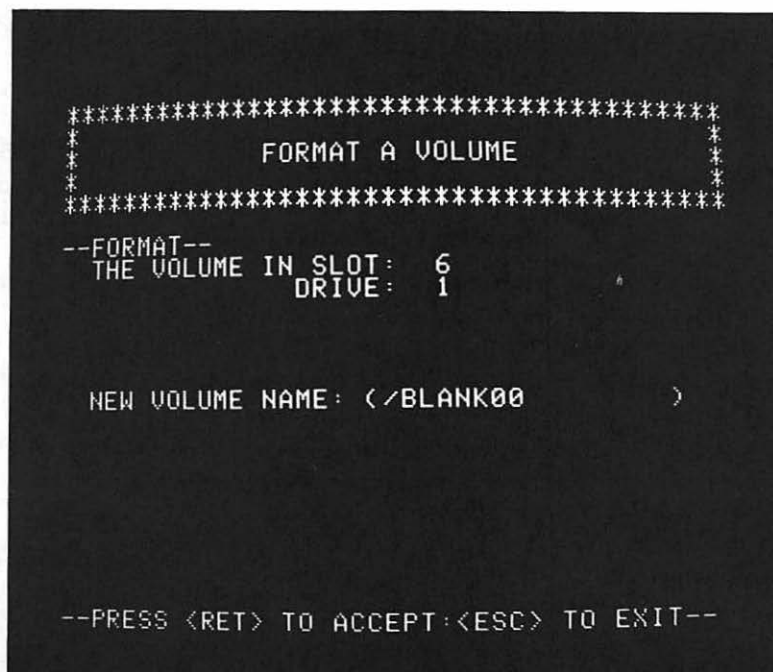


Figure 1.6. Format data entry screen.

mark (/) acts as the delimiter or separator between names that form the pathname. A pathname is a series of file names, preceded and separated by slashes, that indicates the entire path from the volume directory to the file. Delimiters are used for both command structures and pathnames in ProDOS. A great deal more will be said about paths, pathnames, and command structures in other areas of this book.

Next, notice that a period is entered between the words SCRATCH and DISK. ProDOS does not allow you to enter a blank character as the separator between words. Further, you are allowed to enter only 15 characters for any volume, directory, or file name. This is important to remember because DOS 3.3 allows blanks, special characters, and 30 characters per file name. There are other rules to be followed, but these will be discussed later as they are needed.

*By the way, if you make a mistake anywhere along the line, you may retreat by typing the ESCAPE key. So, you may recover easily from any errors.*

*Note: Remove the boot diskette NOW!*

After entering the volume name, type the RETURN key. When you type that key, your system will format the blank diskette in the disk drive selected. When the formatting is com-

plete, format two more blank diskettes with the names /PRODOS and /EXAMPLES. Then you should return to the FILER main menu screen by using the ESC key. Once you have the main menu back you will want to select the F (FILES COMMANDS) selection option. The main menu is shown again as Figure 1.7.

The F option allows you to work with individual files on a diskette rather than with the diskette as a unit, which was enabled by the volume commands that you have just used. Figure 1.8 shows the secondary screen that you are presented. From the screen, select the C option.

At this point, make sure that you have the /SCRATCH.DISK reinstalled in the boot disk drive. The final screen that you will be presented is the copying files data entry screen. This screen is shown in Figure 1.9.

Notice that this screen shows you the PREFIX currently active and stored in the ProDOS image in memory. It just so happens it is probably the same as the diskette from which the FILER program came.

For this screen you will be entering three different sets of data that transfer the three different program files. The three program files to be transferred are:

/PRODOS/PRODOS	to	/SCRATCH.DISK/PRODOS
/PRODOS/BASIC.SYSTEM	to	/SCRATCH.DISK/BASIC.SYSTEM
/PRODOS/STARTUP	to	/SCRATCH.DISK/STARTUP

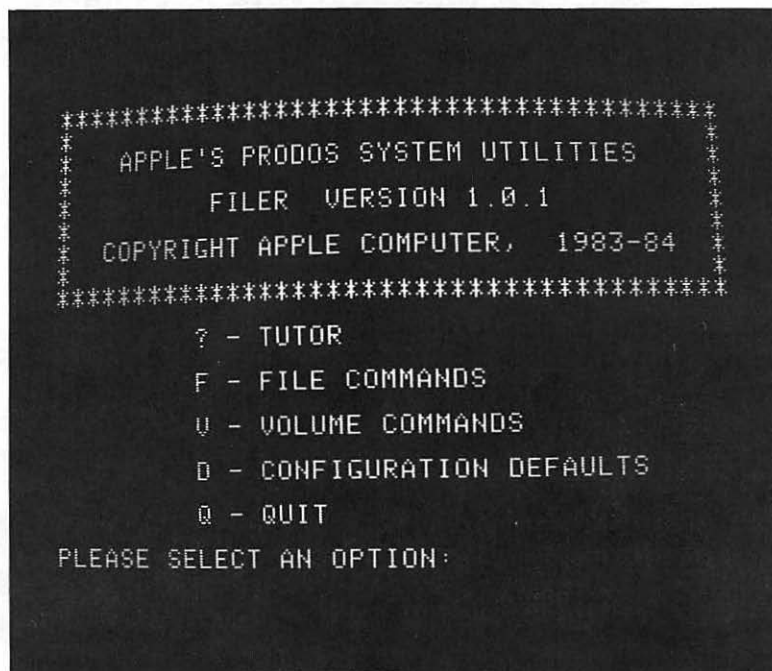


Figure 1.7. FILER main screen.

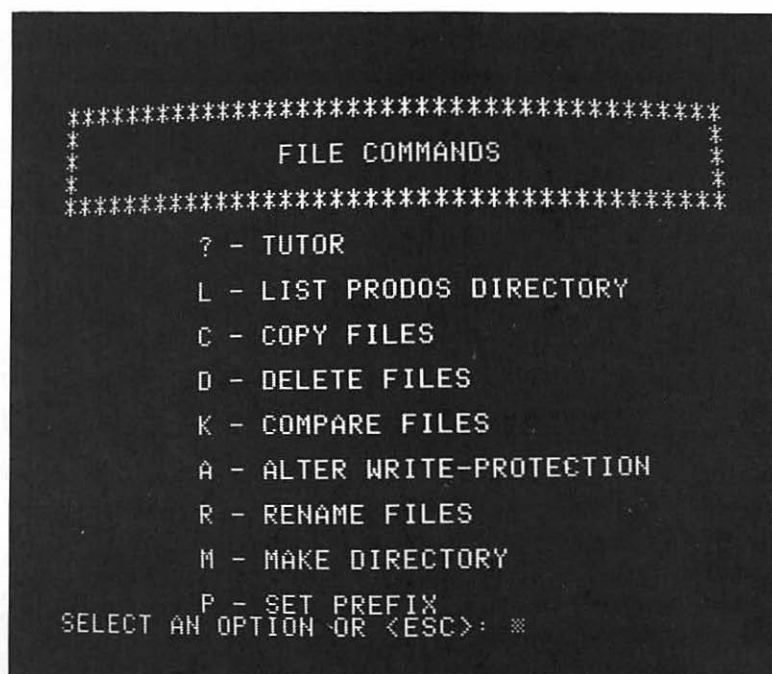


Figure 1.8. File commands screen.

If you are using two disk drives connected to slot 6, then it is recommended that you place the /PRODOS disk in drive 1 and your new SCRATCH.DISK in drive 2. Each time you enter a new set of program names to be transferred, your system will do all of the work.

If you are using one disk drive, the screen will prompt you when it is necessary to exchange diskettes in the disk drive.

## 1.2. MAKE A PRODOS COPY

One of the first things that you should do is to make a copy of the diskettes that came with your ProDOS system. You will want to preserve your original diskettes and use your copy, or backup. If anything adverse should happen, as it has a tendency to do when you are learning a new system, it is better to have it happen to your backup and not to the original. There is a rule, one of Murphy's, I believe, that applies: "If it can happen, it will happen to me." It always does.

Throughout this book and those provided by Apple Computer, Inc., you will be asked to do a number of things with both of the copies you are about to make and also with the new

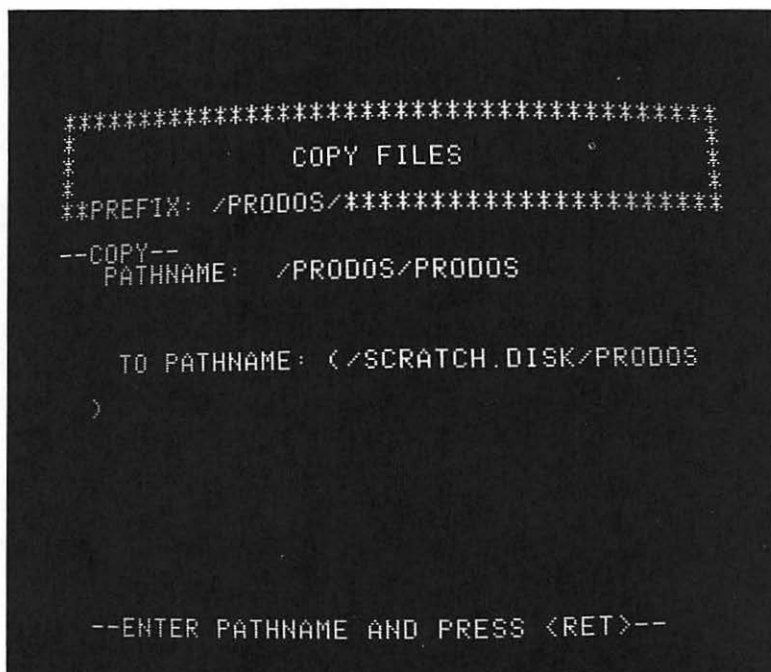


Figure 1.9. Copy files data entry screen.

diskette that you created in Section 1.1.3. There are two disks supplied by Apple Computer, Inc. with ProDOS. These are:

- The /PRODOS disk
- The /EXAMPLES disk

These disks are write-protected. This means that you are unable to write any data or information onto the diskettes. Since some of the programs on these diskettes require that you write information onto the diskettes, let's make copies of these diskettes onto diskettes that are *not* write-protected.

In the last section you formatted and created a diskette called /SCRATCH.DISK. In order to do that, you used the FILER program. In this section you will use that same program again.

So, first get your system powered up and then execute the FILER program. This may be done using



## ]— FILER

The main menu screen should look like Figure 1.10.

From the main menu screen select the V (VOLUME COMMANDS) option. This is done by pressing the V command without using the RETURN key. The secondary screen, shown in Figure 1.11, will again ask you to make another selection. This time select the C (COPY A VOLUME) option.

The third screen, selected by pressing the C key, is the data entry screen. You will use this screen to make copies of both of the original diskettes that came with your system. This screen is shown in Figure 1.12.

When you are finished, you have made copies of all the diskettes that came with your ProDOS system and created the SCRATCH.DISK for your own programs. It is recommended that you label each of these, identifying them with the current date, the fact that they are ProDOS diskettes, and the name of their volume, such as /PRODOS or /SCRATCH.DISK.

Since you probably have a number of DOS 3.3 diskettes, you should start a new filing system just for the ProDOS diskettes. Isn't that wonderful! Now, you have two sets of diskette filing containers.

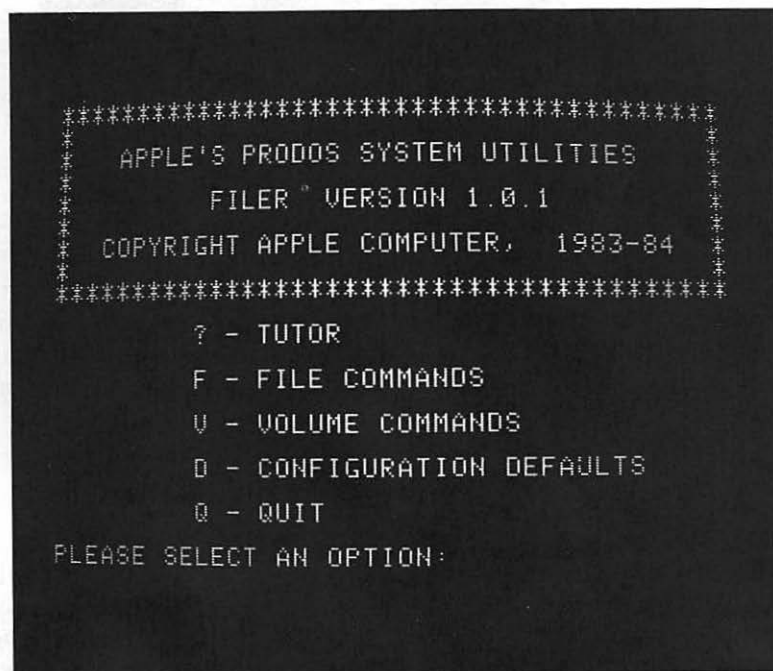


Figure 1.10. FILER main screen.

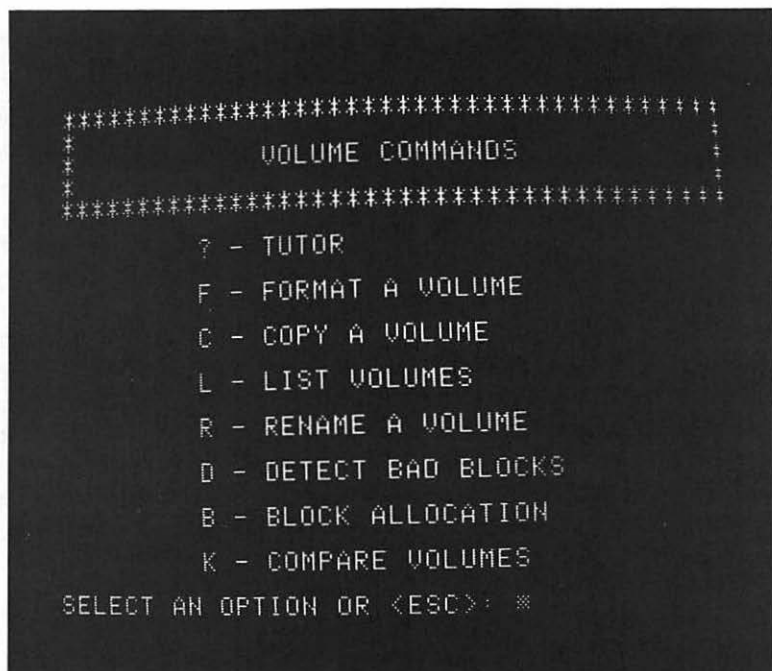


Figure 1.11. Volume commands screen.

### 1.3. PRODOS COMMANDS DESCRIBED

In this section, you will be given the general form for the description of each command that is described in this book. Normally, the general form of a command is known as the syntax required for the command. A command's syntax for ProDOS looks generally like this:

command [pn] [,S#] [,D#]

All of the command coding requirements shown throughout will be within boxes, as above. The word “command” will be used to mean any of the legal ProDOS commands available to you to use while communicating with peripheral storage devices and the outside world. The elements of the command's syntax that are in the square brackets—[pn], [,S#], and [,D#]—are called the command's options. There are, of course, many other possible options available. Each will be shown and discussed in detail as it is needed. If the square brackets are not present around a command option, that option is required to be used in the command.

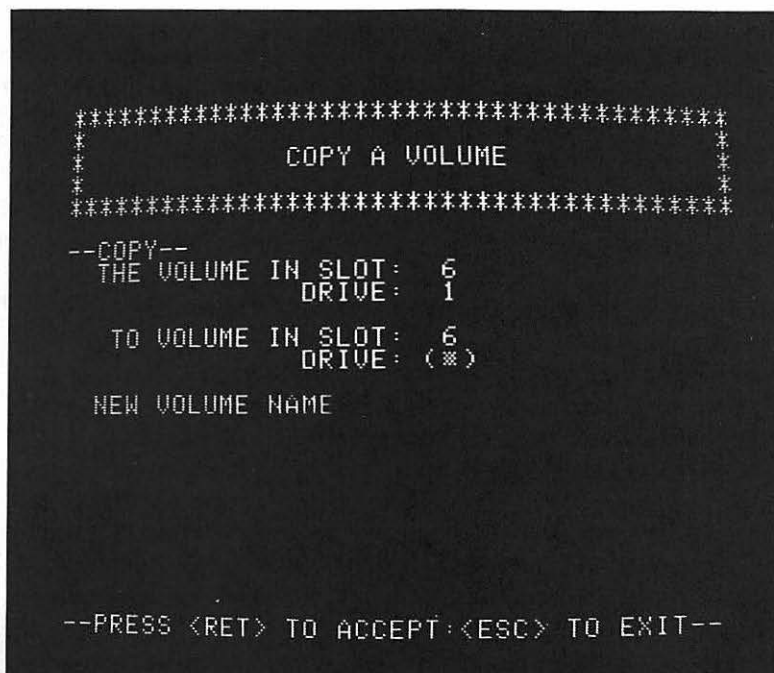


Figure 1.12. COPY A VOLUME display screen.

Notice that the options may have a comma (,) preceding the option. The comma (,) is used as the delimiter between command options like the slash mark (/) delimiter is used in pathnames. The comma is required.

The next required entry after the comma is the command option being entered. These command options *must* be entered as capitalized alphabetic characters.

The options shown for the command general form are known as the pathname, slot number, and drive number, respectively. When they are all taken together, the options form the name and storage location of the file that is to be accessed. Any file may be accessed on any diskette by specifying only the pathname option, only the slot-drive options, or both the pathname and the slot-drive options.

A detailed discussion of pathnames is in Section 2.2.3. The specifying of the slot-drive options is still provided for the purpose of maintaining compatibility with DOS 3.3. Further, the slot-drive option capability allows for additional command control.

An example is given here to give you a feel for how to use the general form of a ProDOS command. Power up your system using the procedure outlined in the next section with the /PRODOS backup diskette in drive 1. When you have the prompt character and the cursor on the video screen, execute the command:

```
JCAT /PRODOS,S6,D1
```

This command tells the operating system to display the files contained on the /PRODOS diskette located in slot 6, drive 1. Pay close attention to the typed form of this command. The CAT command, new in ProDOS, is discussed in Section 3.1. I have not used any blank spaces within the command. Blank spaces are allowed; ProDOS is forgiving in the case of blanks except for commands and pathnames.

*Notice that the square brackets are not used in the writing of a command. The use of the square brackets is strictly for the purpose of separating options within a command.*

The characters within the square brackets perform specific functions in each of the options when entered.

These are:

- The comma (,) separates that option from its predecessor.
- The capital letter identifies the option.

These are:

S— slot	A— address	R— record	\$— hexadecimal
D— drive	B— byte	P— position	@— at line
F— field	T— type file	L— length	
E— end address			

- The characters or numbers following the capital letter represent the value assigned that option.
- The pound sign character (#) signifies the requirement to enter an integer numeric value.
- The dollar sign character (\$) signifies the requirement to enter a hexadecimal address or number.

Even though provision is made for the use and entry of hexadecimal numbers, decimal equivalents may be used. Therefore, you are not required to use hexadecimal notation. However, when using hexadecimal notation, use the dollar sign in front of the hexadecimal number. For example, 255 in decimal notation is \$FF in hexadecimal notation. See Appendix J for a decimal-hexadecimal conversion table.

## 1.4. STARTING PRODOS

If your computer system is already turned on, turn it off completely. Wait for at least 30 seconds before you restart your system. This is to let any and all stray currents and capacitances bleed off the system. Besides, it is hard on the power supply to reactivate power immediately after a shutdown. In order to start up your system with ProDOS active, you should use the following procedure:

1. Turn on your TV or monitor.
2. Turn on your ProFile<sup>™</sup> hard disk. (See Appendix F.5, if applicable.)

3. Wait for a steady READY light on ProFile. (See Appendix F.5, if applicable.)
4. Place the ProDOS diskette into disk drive 1.
5. Reach around to the left rear of your computer and turn it on.
6. The video screen will show you something like Figure 1.14.

Figure 1.13. Startup procedure.

The next two sections will describe various ways of starting up (powering up) your system.

#### 1.4.1. From a cold start

Turning on your Apple computer causes the system to attempt to read information from disk drive 1 installed in a numbered slot (usually slot 6). When the screen of Figure 1.14 is displayed, you know that the PRODOS program is now resident in memory and has been executed. Any diskette that contains the PRODOS program will display the screen in Figure 1.14 when ProDOS is activated.

```

      APPLE ][
      PRODOS 1.0    1-OCT-1983
  
```

COPYRIGHT APPLE COMPUTER, INC., 1983

Figure 1.14. ProDOS title screen.

The screen will clear in a few seconds, and the Applesoft II BASIC prompt character will be placed in the upper left hand corner of the video screen. A few seconds later your screen will again fill, giving you a summary of your system's configuration and the peripheral cards installed in the expansion slots. You are seeing the results of running the STARTUP program. See Figure 1.15.

This second startup screen contains a lot of information. First of all, you are told that you are using version 1.0 of the ProDOS operating system. If you have any problems with your ProDOS, it is a good idea to remember the version number of your operating system.

The volume name of the diskette is /PRODOS. Further, you are told that you are using the Apple IIe with 128K of memory and Applesoft II BASIC in ROM.

The last set of information you are given is a list of the peripheral circuit boards and their slot assignments. Please remember that your particular second screen may be different because your system may be configured differently. Also remember ProDOS requires that you have at least 64K of memory in your machine. This was outlined in Section 1.1.1.

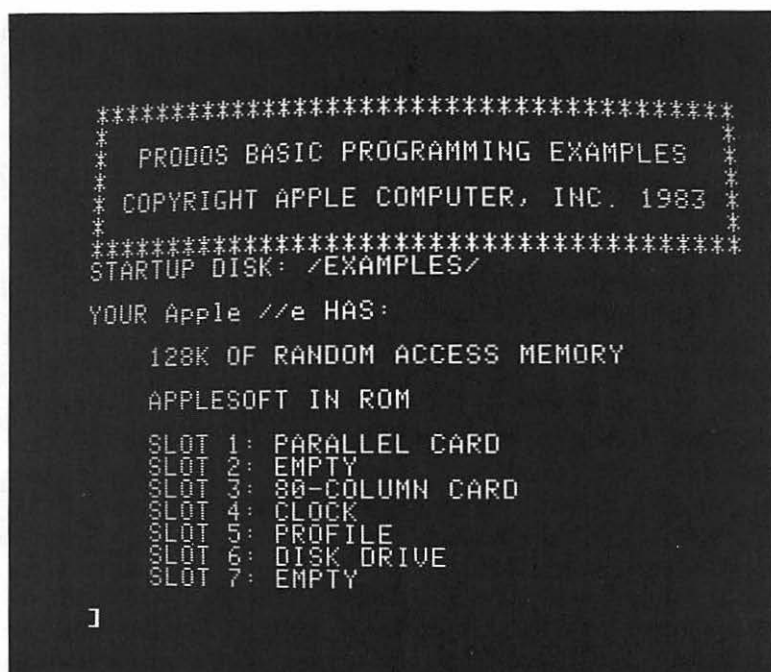


Figure 1.15. System configuration screen.

You now have ProDOS and an Applesoft II BASIC program, called STARTUP, that was stored on the boot diskette, memory resident. Additionally, you have 23 new ProDOS commands that can be used within an Applesoft II BASIC program. These commands look a lot like the Applesoft II BASIC commands and instructions, but they act totally differently. Further, they do not necessarily follow the same rules of construction or execution.

#### 1.4.2. From other ways

If for some reason you find yourself in the monitor program (signified by the \* prompt character), you could try the command:

**\*6 (CTRL-P)**

This means type a 6 on the upper row of the keyboard. Follow that by typing the CTRL and the P simultaneously. Then release both keys at the same time. This should give you a restart of the computer without having to turn it off.

*If the disk controller card is in a slot other than 6, use that slot number instead of 6.*

Another way to restart your system is to issue:

JPR# 6

from the immediate mode. This will cause your system to restart almost as if it were a cold startup.

If you are using an Apple IIe, you have an additional way to restart your machine. You may press the OPEN APPLE, CTRL, and RESET keys simultaneously and then release them simultaneously.

The only item of information that is necessary to remember is that any program or data in memory will be lost. So please be careful.

## 1.5. THE HELP COMMAND

One of the more interesting and helpful capabilities of ProDOS is the HELP command. Has there ever been a time when you couldn't quite remember exactly how a particular command needed to be written? I'd hate to tell you how many times that has happened to me. Now there is help for us: a new command in ProDOS called HELP.

In order to use this command while using ProDOS you can type the following from the immediate mode:

J— /EXAMPLES/HELP

with the /EXAMPLES diskette in one of your disk drives. On your diskettes, the HELP capability may be on another diskette. What this does is to activate the HELP and HELPSCREENS files. Once you have activated the HELP capability, it will remain in memory until you turn off your computer system or use some other system program, such as FILER or CONVERT.

Once the HELP capability has been activated, you may get help with the writing of all ProDOS commands. The syntax for this is:

JHELP command
---------------

where the word "command" represents one of the ProDOS commands. If, however, you just type:

JHELP

you will be shown a video screen like the one in Figure 1.16.

The column on the left of the screen lists command uses by groupings. The column on the right lists those words that may be entered in place of the word "command" when invoking the HELP capability.

In order for the HELP and HELPSCREENS to work, both of them must be on the same diskette installed in a disk drive. You can move these files from their original diskette to your

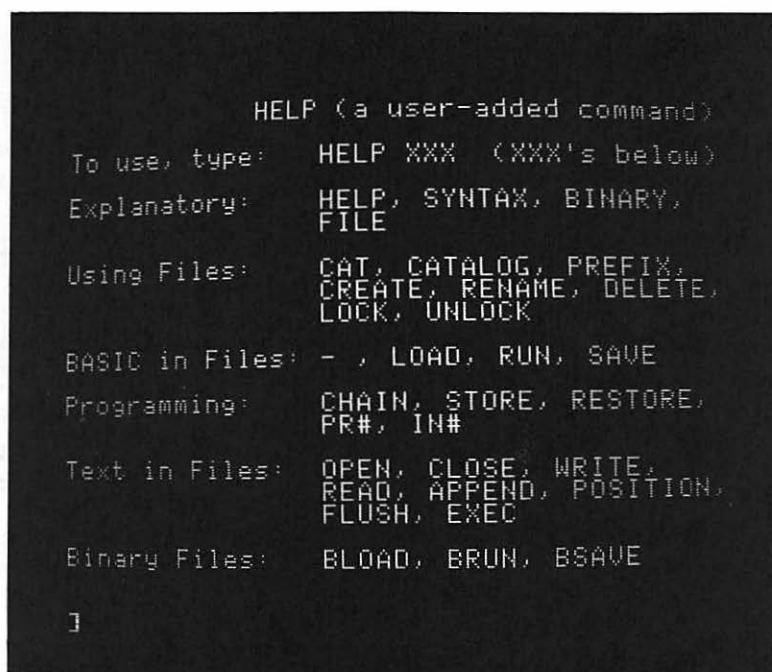


Figure 1.16. HELP selection screen.

SCRATCH.DISK using the FILER program the same way you previously moved other programs. Refer to Sections 1.1.3 and 1.1.4 in this chapter or to Chapter 8.

If you have invoked the HELP capability and later in the session you do not need it anymore, then type:

JNOHELP

to remove the HELP program. Notice that the NOHELP command has no options. Once HELP has been deactivated using the NOHELP command, the HELP command will no longer work and cannot be reinvoked without rebooting the system.

## 1.6. PRODOS AND DOS

There are probably more programs written for the Apple computers than for any other machine in the marketplace today. If your library of programs is like mine, it would be a real



shame not to be able to move some programs into the new ProDOS environment. In this section the differences between ProDOS and DOS 3.3 will be explained.

When you start up your system with a ProDOS diskette in Disk II, the /PRODOS program is placed into memory and executed. This allows you to read from and write to all diskettes installed in all disk drives made for Apple computers. In comparison, when you start up your system with a DOS 3.3 diskette in Disk II, the DOS 3.3 program is placed into memory and executed. This allows you to read from and write to all diskettes installed in Disk II drives only.

The information that ProDOS stores on a diskette cannot be read by DOS 3.3, just as the information the DOS 3.3 stores on a diskette cannot be read by ProDOS. Fortunately, the program (CONVERT) on the /EXAMPLES diskette will perform the conversion from one format to the other. This is similar to the function of the MUFFIN program on the DOS 3.3 SYSTEM MASTER diskette.

What all of this means is that your existing DOS 3.3 programs are not able to use a diskette other than a Disk II format until you convert them from the DOS 3.3 format to the ProDOS format.

As a general rule of thumb, the programs you write can be converted to ProDOS. The programs you purchase from commercial software developers will not necessarily convert to ProDOS. If, however, a program uses only standard Applesoft II BASIC instructions and DOS 3.3 commands, that program is a good candidate for conversion.

If programs have tricky PEEKS and POKES or use any machine-language routines, they may not work correctly even after conversion. If DOS 3.3 file names have blanks, special characters, or are over 15 characters long, your programs also may not work. Even with all of these "ifs," it is worth doing the conversion. Converting programs from DOS 3.3 to ProDOS is explained in Chapter 8.

## SUMMARY

This chapter has given you a first look into ProDOS and how to perform some very basic file manipulations, including making copies and making your own SCRATCH.DISK.

A number of terminology conventions were introduced in this chapter. This was done to provide a common means of communications.

The first item was the introduction to your computer system from a logical point of view, which is somewhat different than the normal hardware look at a computer. The relationship between the various layers of active software and the functions performed at each level were discussed.

You were given the hardware requirements for installing and using ProDOS on your particular system. Each version of Apple II computers was shown.

The diskette contents that came with the new ProDOS were discussed briefly and shown.

You were shown how to make copies of your original ProDOS diskettes and how to create a working diskette called SCRATCH.DISK of your own using the FILER program.

The general form of all of the ProDOS commands was explained so that you will be able to understand how to read and write the ProDOS commands. Many of the options for commands were shown along with the first detailed look at ProDOS commands.

You were also given various ways of powering up or restarting your system and having ProDOS installed as the operating system.

A number of differences between ProDOS and DOS 3.3 were discussed as applicable. This gives you a way to make the transition easily.

Finally, the HELP and NOHELP capability were discussed. These are very handy added capabilities that should be very valuable to you as you develop programs requiring ProDOS commands.

The commands introduced in this chapter were:

HELP	e.g.,	HELP	Immediate mode only
		HELP PREFIX	
NOHELP	e.g.,	NOHELP	Immediate mode only

## QUESTIONS

1. Describe how you may power up or restart your particular system. Describe other methods available.
2. Describe the layers of operating software and their relationships to each other when you use your computer system.
3. Describe in detail what is needed to install and run ProDOS.
4. Describe how to prepare a diskette to receive files.
5. Describe how to copy files and volumes.
6. Describe the syntactic form for all ProDOS commands. Describe how the options operate.
7. Describe how to use the HELP file. What can this file do for you?
8. Describe the differences between the INIT command in DOS 3.3 and the formatting of a volume using the FILER program.
9. What are the results of not having an operating system resident on a formatted diskette?

## 2. PRODOS FILES AND COMMANDS

*If you command wisely,  
you'll be obeyed cheerfully.*

*Thomas Fuller, 1732*

### 2.0. OVERVIEW

The relations among terms such as volume, pathname, filename, and files will be discussed in this chapter. How ProDOS arranges files on the diskette, how files are named, and what terminology is required to refer to files are explained.

The term *file* will refer to any file stored on a diskette, regardless of its type. A file may be an Applesoft II BASIC program file, a binary file, a random access file, or any other named file, stored on a diskette.

The relationship between directories and file names will be shown. How ProDOS works is the subject of Section 2.1. This section explains what is required and how to arrange the required programs on a diskette.

The minimal composition of a startup diskette and the programs involved are explained. Each of the programs required by a startup diskette, and its functions, is discussed.

The logical arrangements of volumes, directories, and files are shown and explained. The rules for directories and volumes are shown and compared to those for DOS 3.3. This provides for any easy transition to ProDOS.

The requirements for naming files are explained in detail alongside a comparison to DOS 3.3 file naming. Legal and illegal file names are given. The creation and uses of pathnames and prefixes are discussed in detail.

This chapter will introduce you to your first and probably most often used ProDOS commands and show how they may be used in programs and from the immediate mode.

The commands introduced in this chapter are shown in the box below.

CAT	e.g.,	CAT CAT /PRODOS	Immediate and deferred
CATALOG	e.g.,	CATALOG CATALOG	Immediate and deferred
PREFIX	e.g.,	PREFIX /PRODOS/STATES	Immediate and deferred
RUN	e.g.,	RUN /PRODOS/DUMMY RUN	Immediate and deferred
—	e.g.,	— DUMMY	Immediate mode only
LOAD	e.g.,	LOAD /PRODOS/VIEW	Immediate and deferred
SAVE	e.g.,	SAVE /MY.DISK/DEMO	Immediate and deferred

## 2.1. HOW PRODOS WORKS

This section gives you an overview of ProDOS. ProDOS is an operating system that allows you to manage many of the resources available to the Apple II computers. It functions primarily as a disk-based set of programs that help you operate your system. However, it also

will handle interrupts and perform some memory management. ProDOS also will mark files with a date and time, if you have had a clock/calendar card installed.

All ProDOS startup diskettes will normally have certain files in common.

These are:

- PRODOS
- BASIC.SYSTEM
- STARTUP

Some of these were mentioned in Chapter 1. In this section each of these will be discussed in more detail.

The PRODOS file contains the ProDOS operating system program. This program performs the communications required between other system programs, Applesoft II BASIC programs, and computer hardware items.

The file BASIC.SYSTEM is a system program that will communicate between the system user and the operating system. A ProDOS system program, such as the BASIC.SYSTEM, FILER, or CONVERT programs supplied when you purchased ProDOS, is an assembly-language program. Each program will accept commands from an operator, check commands for validity, and finally take the appropriate action.

The STARTUP program is the first Applesoft II BASIC program that is executed immediately after the operating system has finished booting and your system is ready to perform useful work, such as Phasor Pilot-Maze. In DOS 3.3 you were allowed to define the boot program to be executed. With SOS 1.1 for the Apple III, the boot program name is required to be HELLO. With ProDOS the name STARTUP is required.

Memory in Apple II computers is divided into 256-byte segments. Each of these segments is considered a page of memory. ProDOS treats memory in your computer in the same manner. For each 256-byte segment used, ProDOS will represent that page by setting a bit to 1 in the system bit map memory area.

When ProDOS is initialized, all memory used is marked used (set to 1) in the system bit map. As ProDOS runs, it marks each new page used by setting the appropriate bit in the system bit map. When a page is released, then that bit in the system bit map is reset, changed to 0.

### **2.1.1. A startup diskette**

ProDOS is able to support and communicate with many different types of disk drives. The type of disk drive, whether full size, half-height, hard disk, or mini-floppy, and the particular physical slot location of the disk drive need not be known by the system program. Instead, the machine-language interface (MLI) has the capability to take care of the interfacing details. The MLI is discussed in Chapter 9.

In Chapter 1 you created a startup diskette called SCRATCH.DISK. The diskettes inserted in the physical disk drives are known as volumes and are identified by names known as volume names. Volumes and volume names are discussed in Section 2.2.

Today, almost every Apple II family computer system will have at least one disk drive. That disk drive is known as the system's startup or boot drive. This drive will normally be con-

nected to the drive 1 connection on the disk controller card that is normally installed in slot 6. For the discussion that follows, it will be assumed that:

1. The system startup drive is connected to slot 6, drive 1.
2. Your system has the AUTOSTART ROM chip installed.
3. Your system is configured to accept ProDOS as outlined in Chapter 1.

When you put a diskette, the boot or startup diskette, into the system startup drive (S6,D1) and power up your system, the AUTOSTART ROM chip program first executes. This forces the system to boot up from the startup diskette by installing into memory and executing the ProDOS operating system and by preparing your system to perform work.

The diskettes that cause your system to start up are known as startup diskettes. A startup diskette must contain all of the information needed to bring a program from the diskette into your system's memory and to commence executing the program. A ProDOS startup diskette holds all of the information needed to bring the operating system into memory and start it running.

Any ProDOS diskette can be made into a startup diskette by placing the correct files on that diskette. You did that in Chapter 1 when you created the SCRATCH.DISK. Then, you did it almost blindly, without explanation. In the next few paragraphs is some explanation. At power up time, a copy of the ProDOS program is automatically transferred from the diskette into memory and executed.

A ProDOS startup diskette has the following characteristics:

- It was formatted using the ProDOS FILER program.
- It has the PRODOS program in its volume directory.
- It has the BASIC.SYSTEM program in its volume directory.
- It has an Applesoft II BASIC program called STARTUP in its volume directory.

### 2.1.2. The PRODOS program

The PRODOS program is just what the name implies. It is a set of machine-language routines that provide the interface to any disk drive manufactured by Apple Computer, Inc. for the Apple II computers. PRODOS is an operating system program that allows you to manage many of the resources available to the Apple II computers, plus handle interrupts and simple memory management. In addition to all of the above, PRODOS allows you to interface your own particular routines and additions to ProDOS.

PRODOS has a number of major modules. These are:

- System program
  - receives user commands
- External device routines
- Command Dispatcher
- Machine Language Interface

Block File Manager  
     disk driver routine  
     clock/calendar routine  
 Interrupt Receiver/Dispatcher  
     interrupt handling routines

### 2.1.3. The BASIC.SYSTEM program

This part of the ProDOS operating system contains all of the operating system commands and error routines that are supported by ProDOS. BASIC.SYSTEM allows the user and the operating system to communicate.

In DOS 3.3 the entire operating system is contained in one program. In ProDOS the operating system is divided into two parts. This arrangement allows you the ability to insert, and, or modify your system's operation. PRODOS contains only the most essential parts of the operating system. BASIC.SYSTEM allows you to communicate with disk drives from within Applesoft II BASIC. You may want to install other or different system programs on a diskette. System programs are recognizable in a CATALOG or CAT presentation by the SYS abbreviation in the type column.

When you system boots up, the PRODOS program is loaded into memory and then executed. Then the first system program stored on the diskette is loaded into memory and executed. Therefore, you need not have BASIC.SYSTEM as the next file to be executed. Any file with a name XXXX.SYSTEM (where XXXX may be any combination of letters and numbers that forms a valid name) may be loaded into memory and executed. More will be said about this capability later and in Appendix J.

### 2.1.4. The STARTUP program

This program is an Applesoft II BASIC program that is run by BASIC.SYSTEM when booting is finished. This program is comparable to the HELLO program of DOS 3.3 or SOS 1.1.

The Apple III operating system (SOS 1.1) requires that this program be called HELLO and be written in Apple's Business BASIC. The Apple II computers operating in DOS 3.3 allow you to name this program anything you desire, provided it is memory resident when that diskette is initialized. It may be written in either of the two BASIC languages. ProDOS requires that this program be written in Applesoft II BASIC and be called STARTUP.

If BASIC.SYSTEM does not find an Applesoft II BASIC program named STARTUP, the following message is then displayed:

PRODOS 1.0      ©1983 APPLE COMPUTER

and you are left in the immediate mode with the Applesoft II BASIC prompt character and cursor. This means that the booting process is complete and it is up to you to command your system for the next evolution.

## 2.2. VOLUMES AND FILES

A file is the basic informational storage unit. Any file may contain any set of information such as names, numbers, letters, pictures, Applesoft II BASIC programs, lists, machine-language programs, or graphs.

A file may be defined as a collection of related information stored on some medium under a shared name. When a file is stored on a ProDOS diskette, it is assigned a name and a file type. After a file is stored, access to the information stored in that file is gained through the use of that file's name. The file's type determines the kind and character of the information stored in the file.

When you assign a name to a file, there are a few rules that you must follow. In the following paragraphs, the ProDOS file name conventions will be given, followed immediately with the DOS 3.3 conventions. In this way you will be able to make comparisons and quickly understand the differences.

A ProDOS file name:

- is composed of up to 15 characters.
- must have a capital letter of the alphabet the first character.
- may contain any alphabetic characters (A–Z and a–z).
- may contain any numeric digit (0–9)
- may contain the period character (.).
- automatically converts lowercase characters to uppercase.
- must be unique. This means that no two names are to be exactly the same in one directory.

Files of the same name must be in different directories or on different diskettes.

A DOS 3.3 file name:

- is composed of up to 30 characters.
- must have a capital letter of the alphabet as the first character.
- allows all typable characters except a comma (,).
- must be unique. Files of the same name must be on different diskettes.

Figure 2.1 shows a number of legal and illegal ProDOS file names with comments concerning the illegal file names.

<i>Legal file names</i>	<i>Illegal file names</i>	<i>Comments</i>
TWO.NAMED.FILES	2.NAMED.FILES	Begins with a number
D2	.D2	Begins with a period
A.123.FILE	A 123 FILE	Contains spaces
FUNNY.FACE	FACE,FUNNY	Contains a comma
DISESTABLISH	DISESTABLISHMENT	More than 15 characters



DUMMY.DATA  
Funny.file

DUMMY/DATA  
funny.file

Contains a slash character  
First character not capital

Figure 2.1. ProDOS file names.

There are files of many types, such as program files, text files, binary files, and the most important new type known as the directory file. The directory file is discussed next.

### 2.2.1. The directory

A directory file is like any other file except that it contains only the names, locations, and types of the files in that directory. Figure 2.2 shows two different directories.

This figure shows that the volume directory presently contains four files. These are two additional directories or subdirectories and two other files. The additional subdirectories each contain a number of other files. The diagram also shows that additional files and directories may be added, up to the capacity of the diskette. In fact, ProDOS is able to support a file or file structure of up to 32 megabytes, not on one diskette, however.

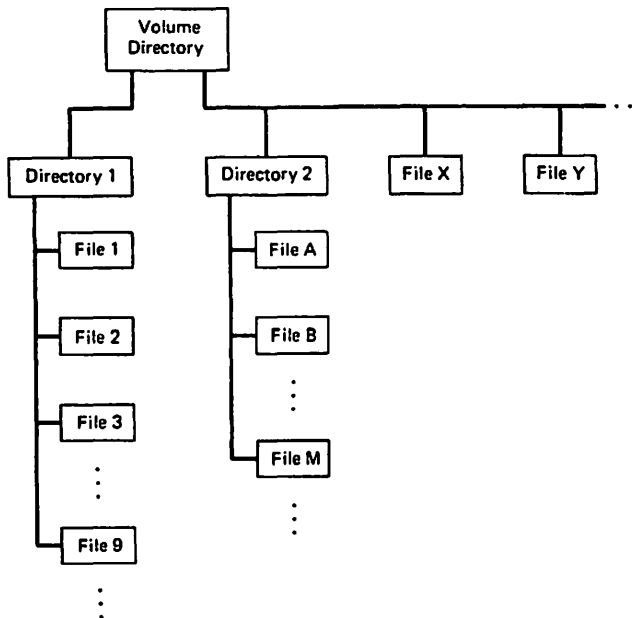


Figure 2.2. Directory file examples.

Notice that the directories may contain any number of files of any type. In fact, one or more of the files in a directory may be another directory, known as a subdirectory. ProDOS allows you to have up to and including 64 levels of directories on any one diskette or other storage medium. You will find that trying to handle effectively more than four or five levels of directories is difficult. It becomes somewhat lengthy to program, to remember, and to type correctly.

In Chapter 1, you formatted a diskette. By the way, a formatted diskette does not contain the operating system. The FILER program was used to place a special directory on the newly formatted diskette. It is called the volume directory and has the name you assigned, known as the volume name. This is the main directory for the entire diskette. The volume directory characteristics are shown in Figure 2.2.

A ProDOS volume directory:

- is on every ProDOS formatted diskette.
- is named when you format a diskette.
- identifies the entire contents of that diskette.
- is the diskette's name.
- may contain up to 51 files.
- may not be created using the CREATE command or the FILER command, Make Directory.
- cannot be removed using the DELETE command.
- cannot be protected using the LOCK command.
- may only be removed by reformatting the diskette.

If you want to see the contents of an entire diskette, it is only necessary to use the volume directory name coupled with the CAT or CATALOG command. These two commands are discussed next.

Place the backup copy of the /PRODOS diskette in drive 1 and power up your system using the procedure outlined in Section 1.4. Then try the following commands using your backup copy of the /PRODOS diskette:

**JCAT /PRODOS**

This command will give you the contents of the /PRODOS diskette. In the 40-column format, your screen will look similar to that shown in Figure 2.3a.

Now, if you have an 80-column board in slot 3 or an Apple IIe, try the following after invoking the 80-column format. The 80-column format may be invoked by typing:

**JPR#3**

Now, you use the other cataloging command:

**JCATALOG /PRODOS**

/PRODOS						
NAME	TYPE	BLOCKS	MODIFIED	CREATED	ENDFILE	SUBTYPE
#PRODOS	SYS	31	1-JAN-84 0:00	<NO DATE>	15360	
#BASIC.SYSTEM	SYS	21	15-NOV-83 0:00	<NO DATE>	10240	
#CONVERT	SYS	42	1-NOV-83 0:00	<NO DATE>	20480	
#FILER	SYS	51	1-JAN-84 0:00	<NO DATE>	25600	
#STARTUP	BAS	7	16-NOV-83 15:57	<NO DATE>	2639	
#EDASM.SYSTEM	SYS	9	14-DEC-83 15:32	<NO DATE>	4095	
#EDASM.ED	BIN	17	14-DEC-83 15:32	<NO DATE>	8191	A=\$3000
#EDASM.ASM	BIN	29	14-DEC-83 15:26	<NO DATE>	14079	A=\$3000
#RBOOT	BIN	1	14-DEC-83 15:35	<NO DATE>	410	A=\$0210
#RLOAD	BIN	3	14-DEC-83 15:36	<NO DATE>	661	A=\$0000
#EXERCISER	SYS	16	17-OCT-83 0:00	<NO DATE>	7385	
#BUGBYTER	BIN	16	15-DEC-83 0:00	<NO DATE>	7177	A=\$2000
#FORMATTER	TXT	3	20-DEC-83 0:00	20-DEC-83 0:00	764	R= 0
#APA	BIN	10	11-NOV-83 7:37	<NO DATE>	4688	A=\$2000
#BYE	BAS	1	16-NOV-83 15:55	<NO DATE>	150	
#ABLE	BIN	4	21-NOV-83 11:10	<NO DATE>	1000	A=\$2000
BLOCKS FREE:	12	BLOCKS USED:	268	TOTAL BLOCKS:	280	

Figure 2.3a. CAT files on /PRODOS.

/PRODOS				
NAME	TYPE	BLOCKS	MODIFIED	
#PRODOS	SYS	31	1-JAN-84	
#BASIC.SYSTEM	SYS	21	15-NOV-83	
#CONVERT	SYS	42	1-NOV-83	
#FILER	SYS	51	1-JAN-84	
#STARTUP	BAS	7	16-NOV-83	
#EDASM.SYSTEM	SYS	9	14-DEC-83	
#EDASM.ED	BIN	17	14-DEC-83	
#EDASM.ASM	BIN	29	14-DEC-83	
#RBOOT	BIN	1	14-DEC-83	
#RLOAD	BIN	3	14-DEC-83	
#EXERCISER	SYS	16	17-OCT-83	
#BUGBYTER	BIN	16	15-DEC-83	
#FORMATTER	TXT	3	20-DEC-83	
#APA	BIN	10	11-NOV-83	
#BYE	BAS	1	16-NOV-83	
#ABLE	BIN	4	21-NOV-83	
BLOCKS FREE:	12	BLOCKS USED:	268	

Figure 2.3b. CATALOG files on /PRODOS.

In this case, notice that the video screen contains more information. This is shown in Figure 2.3b.

There is a detailed discussion of the meaning of each of the columns and the specific differences between the CAT and CATALOG commands in Section 3.1. For now, just realize that there are differences in the commands and different information is presented to the video screen depending upon which command is used.

Notice that some of the files are directories. These contain the names and addresses of other files on the diskette. This can be immediately recognized by the DIR abbreviation in the TYPE column to the right of the file name presented on the video screen.

The program that follows reads a directory and displays the contents of that directory on your video screen in either a 40-column or an 80-column format. It assumes that your 80-column text card is installed in slot 3 and may be activated in the normal way.

]NEW

]LIST

```

1  REM ***** READ.DIRECTORY *****
2  REM *
3  REM * WRITTEN BY:JL CAMPBELL
4  REM *      DATED:10/10/1983
5  REM *
6  REM * COMPUTER:APPLE II+ & IIE
7  REM * LANGUAGE:APPLESOFT II
8  REM *
9  REM *****
10 TEXT : CLEAR : HOME : POKE 216,0:D$ = CHR$(4)
15 PRINT D$;"FRE"
20 PRINT CHR$(27);CHR$(17);: REM FORCE 40-COLS
25 FOR I = 1 TO 39: PRINT "*";: NEXT I: PRINT
30 PRINT "*";: HTAB 39: PRINT "*"
35 PRINT "*";: HTAB 14: PRINT "READ CATALOG";: HTAB 39: PRINT "*"
40 PRINT "*";: HTAB 39: PRINT "*"
45 FOR I = 1 TO 39: PRINT "*";: NEXT I: PRINT
50 VTAB 7: CALL - 958: INPUT "DIRECTORY NAME = ";PR$
52 IF LEFT$(PR$,1) = "/" THEN 56
54 PR$ = "/" + PR$
56 IF RIGHT$(PR$,1) < > "/" THEN 60
58 PR$ = LEFT$(PR$, LEN (PR$) - 1)
60 VTAB 9: CALL - 958: INPUT "CHOOSE: 1=40-COLS 2=80-COLS ";N
65 IF N = 1 OR N = 2 THEN 80
70 VTAB 22: HTAB 13: INVERSE : PRINT "ILLEGAL ENTRY!";
  CHR$(7): NORMAL
75 FOR I = 1 TO 1000: NEXT I: GOTO 60
80 IF N = 1 THEN W = 39
85 IF N = 2 THEN W = 79: PRINT D$;"PR#3": REM ACTIVATE 80-COLS
90 HOME
100 PRINT D$;"OPEN ";PR$;"",TDIR"
120 PRINT D$;"READ ";PR$

```

```

130 INPUT N$: PRINT LEFT$ (N$,W): REM READ NAME
140 INPUT T$: PRINT LEFT$ (T$,W): REM READ TITLE
150 INPUT L$: PRINT L3$: REM READ BLANK LINE
160 INPUT NF$: PRINT LEFT$ (NF$,W): REM READ FILE NAMES
170 IF NF$ < > " THEN GOTO 160
180 INPUT B$: PRINT LEFT$ (B$,W): REM READ BLOCK COUNT
190 PRINT D$;"CLOSE ";PR$
200 END

```

A line-by-line explanation of the above program follows.

<i>Line</i>	<i>Description</i>
Line 1-9	Identify the program.
Line 10	Set up a clean machine. TEXT means to place your machine in the text mode and place the cursor at the bottom of the screen. CLEAR means to clear all variables and more importantly the processor stack. HOME means to place the cursor at the top of the screen and clear the entire screen. POKE 216,0 means to turn off the disk operating system error flag. D\$ is assigned the disk operating system character.
Line 15	Perform a quick ProDOS garbage collection.
Line 20	Print those characters that sets your machine to a 40-column presentation.
Line 25	Print a line of asterisks at the top of the screen.
Line 30	Print asterisks at the left and right edge of the screen.
Line 35	Print screen title.
Line 40	Print asterisks at the left and right edge of the screen.
Line 45	Print a line of asterisks on the screen.
Line 50	Position cursor, clear the screen, and ask operator to enter the directory name.
Line 52	Test the left of the pathname for a slash, "/".
Line 54	If slash is not present then affix slash to the front of the name.
Line 56	Test the right end of the directory name for a slash.
Line 58	Take the right end slash off the name.
Line 60	Position cursor to line 9. Clear the screen. Ask operator for the presentation required.
Line 65	Test the response for a valid answer.
Line 70	Error message if response is not correct.
Line 75	General delay loop for error message presentation.
Line 80	If 40-column presentation, then set the width, W, to 39.
Line 85	If 80-column presentation, then set the width, W, to 79. Then turn on the 80-column card in slot 3.
Line 90	Clear the screen.

Line 100	Open the directory file specified.
Line 120	Read the file specified.
Line 130	Input the name of the directory and print it on the screen.
Line 140	Input the column titles that are to be presented.
Line 150	Input the blank line.
Line 160	Input the first file name in the directory specified.
Line 170	Test for a null string.
	If false, loop back and read another file name and data.
Line 180	Input the blocks used and free count.
Line 190	Close the specified file.
Line 200	End the program.

Note: This program assumes you have an 80-column card installed in slot 3.  
The next section talks about a pathname and how it is used.

### 2.2.2. The pathname

ProDOS must know how to find any file that you want to retrieve from a diskette. In order to do this, you must define to ProDOS “the yellow brick road” to follow from the diskette’s volume directory to the storage location of the file you want. This road is called the “pathname.” The pathname concept is not supported in DOS 3.3.

The pathname defines to ProDOS how to proceed from the volume directory to the file being retrieved. For example, I need to know the PROFIT information from the subdirectory LEMONADE.STAND in the volume directory MY. The pathname would then be:

**/MY/LEMONADE.STAND/PROFIT**

Now that you know how a pathname looks, let me define how it is composed.  
A ProDOS pathname:

- is a series of file names, preceded and separated by slashes.
- has a slash as the first character in the pathname.
- has a volume directory file name as its first entry.
- is less than 65 characters long, including slashes and all subdirectory names and file name.

These rules are shown diagrammatically in Figure 2.4.

Next, let us look at a typical set of files that might be stored on a diskette. The file storage structure is shown diagrammatically in Figure 2.5.

From the diskette diagram in Figure 2.5 you can see how files are organized on a diskette. The volume directory name is MY. This directory contains three files, BANK, HOME, and CREDIT.CARDS. The files BANK and HOME are subdirectories. The BANK subdirectory contains two files named NOTES and CASH. The NOTES file is another subdirectory that contains the files DUE and PAID. The HOME subdirectory also contains two files named STOCKS and BONDS. The pathnames to each of the files on the MY diskette are:

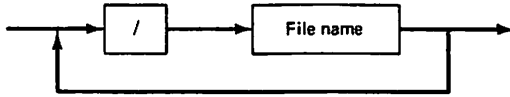


Figure 2.4. ProDOS pathname structure.

```

/MY/BANK/NOTES/DUE
/MY/BANK/NOTES/PAID
/MY/BANK/CASH
/MY/HOME/STOCK
/MY/HOME/BONDS
/MY/CREDIT.CARDS

```

Another view of the pathnames to the diskette contents are shown below. Notice that the directories and subdirectories are shown with a slash character preceding the name, and no slash character is shown for file names. The reason for this is because ProDOS allows you to set a partial pathname known as the PREFIX. The PREFIX is discussed in Section 2.2.3. With a properly assigned PREFIX, you need only refer to a file by its name.

```

/MY
 /BANK
  /NOTES
   DUE

```

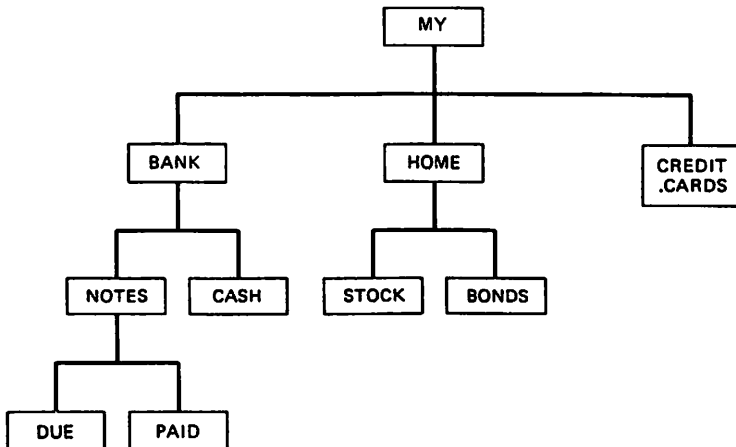


Figure 2.5. Typical disk file storage.

```

    PAID
    CASH
  /HOME
    STOCKS
    BONDS
  /CREDIT.CARDS

```

Now you know how to proceed from the volume directory to a file and how a pathname is constructed. It would really be nice if there were a shorthand way to set or save a pathname or partial pathname. Fortunately, there is a way to reduce your typing on the keyboard or reduce the size of your coding requirements. It is called the **PREFIX**. The prefix capability is discussed in the next section.

### 2.2.3. The **PREFIX**

As you store new files on a diskette in different directories or subdirectories, the pathname required to reach those files could get rather long. As pathnames get longer, typing errors are more prone to occur. ProDOS provides you with a means of having to type a pathname or partial pathname once. It is called **PREFIX**. The prefix capability is not supported by DOS 3.3.

By assigning the **PREFIX** variable name to the pathname or to a partial pathname, you can refer to a particular file name without having to type in the entire pathname. A partial pathname is actually only the pathname minus what has been assigned to the **PREFIX**.

The best way to understand the prefix concept is through the use of some simple examples. These are shown in Figure 2.6.

<i>ProDOS must find file</i>	<i>Current PREFIX is</i>	<i>You should type</i>
/DISKNAME/REALLY/LEGAL	/DISKNAME/	REALLY/LEGAL
/DISKNAME/REALLY/LEGAL	/DISKNAME/REALLY/	LEGAL
/MY/BANK/NOTES/DUE	/MY/	BANK/NOTES/DUE
/MY/BANK/NOTES/DUE	/MY/BANK/	NOTES/DUE
/MY/BANK/NOTES/DUE	/MY/BANK/NOTES/	DUE
/MY/HOME/STOCK	/MY/	HOME/STOCK
/MY/HOME/STOCK	/MY/HOME/	STOCK
/MY/HOME/BONDS	/MY/HOME/	BONDS
/MY/CREDIT.CARDS	/MY/	CREDIT.CARDS

Figure 2.6. Prefixes and pathnames.

Notice that some of the examples in Figure 2.6 show the pathnames that were shown diagrammatically in Figure 2.5. The third column entries of Figure 2.6 are partial pathnames.



A full pathname is formed by concatenating the PREFIX with the partial pathname. If a prefix does not match any portion of a pathname, then you should change the PREFIX name.

It is recommended that you thoroughly understand the pathname and partial pathname concept because you will be using them all the time in your programming. The rules for forming a partial pathname are shown below.

A ProDOS partial pathname is:

- a file name, or a series of file names separated by slashes.
- the pathname minus the current prefix.
- less than 65 characters long, including pathnames and slashes.

The PREFIX command is used to set the prefix name. This command is discussed in detail in Section 3.2.

## 2.3. PRODOS AND PROGRAMS

This section describes those ProDOS commands you will need to make use of Applesoft II BASIC programs stored on a diskette. You will find that there are some very interesting added capabilities to these ProDOS commands as compared to DOS 3.3.

The next section introduces you to a new and very handy capability. It is the — (DASH) command.

### 2.3.1. The — (DASH) command

The — (DASH) command is a new feature of ProDOS. This command allows you to bring into memory and run an Applesoft II BASIC program, machine-language routine, an EXEC file, or a system program. In simpler terms, the — (DASH) command allows you to run from the disk any program regardless of which program type is supported by ProDOS. The — (DASH) command is also known as an intelligent RUN command. This command is not supported in DOS 3.3.

To run a program using this capability you issue from the immediate mode the command:

**]— pn [,S#] [,D#]**

Let's look at an example. Suppose that you want to run the FILER program on the USERS.DISK that came with your ProDOS system. This was done in Chapter 1 when you copied diskettes and created a diskette of your own. Place the EXAMPLES in one of your disk drives and then type:

**]— /EXAMPLES/FILER**

This command, when executed, will bring the FILER program into memory and then start execution of the program. The — (DASH) command may be used with any type of program file. The file types are described in Chapter 3.

There is only one caution for using the — (DASH) command with a system program. As the system program is brought into memory, everything else in memory is erased or destroyed. If you are writing an Applesoft II BASIC program, please, make sure that the latest version has been saved before running any system program.

### *Option      Description*

pn	pn is the pathname or partial pathname of the file containing the program you want to run. The file type must be BAS, BIN, TXT, or SYS. Binary files are loaded at the address from which they were saved.
[,S#]	The slot option defines the disk drive slot location.
[,D#]	The drive option defines the disk drive location in a slot.

### *Examples of the — (DASH) command*

— FILER  
 —/USERS.DISK/FILER  
 — EXAMPLES  
 — CONVERT

### **2.3.2. The RUN command**

The RUN command is used for the purpose of loading and executing an Applesoft II BASIC program stored on a diskette. You may use either of the two commands that are shown below:

]RUN pn [,@#] [,S#] [,D#]
]— pn [,S#] [,D#]

The RUN command is described in this section. The — (DASH) command was discussed in the previous section. The RUN command acts the same as in DOS 3.3, except for the new @# option. This newly added option will be discussed in this and other sections of this book.

When you use the RUN command without the @# option, ProDOS will find the program file indicated by the pathname, LOAD the file into memory, and commence execution of the program file from the beginning of the file.

For example, to run the program MATH.DEMO located on your /SCRATCH.DISK diskette, use the command:

**]RUN /SCRATCH.DISK/MATH.DEMO**

When the program is finished, you may restart it by using the command:

**]RUN**

without any options.

You may do this because the RUN command without the file name option present is actually an Applesoft II BASIC command instruction. Therefore, ProDOS will relinquish control of your system in favor of the Applesoft II BASIC language interpreter for further processing. You see, the RUN command without the file name option is not a valid ProDOS command.

<i>Option</i>	<i>Description</i>
<b>pn</b>	pn is the pathname or partial pathname of the file containing the program you want to run. The file must be an Applesoft II BASIC program.
<b>[,@#]</b>	By using this option, an Applesoft II BASIC program will commence execution at the line number specified by the #. If this option is omitted, program execution begins at the lowest numbered line in the program.
<b>[,S#]</b>	The slot option has its normal meaning.
<b>[,D#]</b>	The drive option has its normal meaning.

Let's create a very simple program to illustrate the RUN command. Power up your system with the SCRATCH.DISK in the boot drive. Now enter the following program.

**]NEW**

**]LIST**

```
100 PRINT "LINE 100 EXECUTED"
200 PRINT "LINE 200 EXECUTED"
300 END
```

Now that you have entered this program, save it on your diskette using:

**]SAVE /SCRATCH.DISK/EXAMPLE**

Let's now RUN the program. You should see the following.

```
LINE 100 EXECUTED
LINE 200 EXECUTED
```

Clear the current program from memory by typing:

**JNEW**

The next thing to do is to run the **EXAMPLE** program from the diskette starting at line 200. This is done by typing:

```
JRUN EXAMPLE,@200
```

You should see:

```
LINE 200 EXECUTED
```

printed on the screen.

If you **LIST** the program, you see that the entire program is in memory.

*Examples of the RUN and RUN @# commands*

```
RUN                                * Applesoft II BASIC command
RUN EXAMPLE
RUN EXAMPLE,@100
RUN EXAMPLE,@200,S6,D1
```

### 2.3.3. The LOAD command

If you want to move a copy of an Applesoft II BASIC program from its diskette storage location to the memory of your system, use the **LOAD** command. This command works exactly as it does in DOS 3.3. The syntax of this command is:

```
]LOAD pn [,S#] [,D#]
```

This command is used when you want to examine, modify, or list a program. When you load a program or file into memory, you are actually loading a copy of that program or file into memory. The diskette still contains that program or file unchanged. There is only one caution for loading a program: any previous program in memory is lost. So, be careful.

Once a program is loaded into memory, you may run it by typing **RUN**. What this means is that a program in memory is not identified by a name.

<i>Option</i>	<i>Description</i>
<b>pn</b>	pn is the pathname or partial pathname of the file containing the Applesoft II BASIC program you want to load into memory.
<b>[,S#]</b>	The slot option has its normal meaning.
<b>[,D#]</b>	The drive option has its normal meaning.

*Examples of the LOAD command*

```
LOAD EXAMPLE
LOAD EXAMPLE,S6,D1
LOAD /SCRATCH.DISK/EXAMPLE
LOAD /SCRATCH.DISK/EXAMPLE,S6,D1
```

**2.3.4. The SAVE command**

The SAVE command is for the purpose of transferring the program currently in memory to a program file on a diskette. This command works exactly the same as in DOS 3.3. The syntax for this command is:

]SAVE pn [,S#] [,D#]

When the file is stored on the diskette, it is saved as an Applesoft II BASIC program (BAS). If this file never existed, ProDOS will create the file.

When you save a program file using a program name that already exists on a diskette, that file must be unlocked and be of the same type. This means that you cannot save a text type file of the name DUMMY, for example, to an Applesoft II BASIC program file of the same name.

Every file on a diskette *must* have a unique file name in addition to having the correct data type match for that file name.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn is the pathname or partial pathname of the program file you want to save. If the pn program file already exists, that program must be unlocked.
[,S#]	The slot option has its normal meaning.
[,D#]	The drive option has its normal meaning.

*Examples of the SAVE command*

```
SAVE EXAMPLE
SAVE EXAMPLE,S6,D1
SAVE /SCRATCH.DISK/EXAMPLE
SAVE /SCRATCH.DISK/EXAMPLE,S6,D1
```

**SUMMARY**

The concepts of a volume and its contents, the files and other directories, were discussed. This chapter also showed the relations among a volume, a directory, a subdirectory, and file.

ProDOS and how it works were described in some detail. You were shown how ProDOS manages the resources available.

The requirements for creating a ProDOS startup diskette were discussed. Each of the required programs and their purposes were explained.

The rules for creating file names were outlined and defined. A comparison was made to the rules for creating DOS 3.3 file names.

The pathnames for files and how to use pathnames and partial pathnames were discussed and examples, both legal and illegal, were given. The PREFIX capability and how it relates to pathnames was discussed.

The following figure summarizes the pathname, slot, and drive option combinations available in ProDOS.

[pn]	[,S#]	[,D#]	<i>Pathname desired</i>
—	—	—	See command description
ppn	—	—	pn = prefix + ppn*
ppn	+	+	pn = vn + ppn
ppn	+	—	pn = vn + ppn**
ppn	—	+	pn = vn + ppn***
pn	—	—	pn = pn
pn	+	+	pn = pn

*Notes:*

- + = option used
- = option not used
- pn = pathname
- ppn = partial pathname
- vn = volume name of diskette
- \* = If the prefix is empty, the last value of slot and drive is used.
- \*\* = When only slot is given, drive 1 is assumed
- \*\*\* = When only drive is given, the last slot value used is assumed.

Figure 2.7. Pathname slot-drive summary.

In this chapter you were introduced to the following ProDOS commands:

CAT	e.g., CAT CAT /PRODOS	Immediate and deferred
CATALOG	e.g., CATALOG CATALOG /PRODOS	Immediate and deferred
PREFIX	e.g., PREFIX /PRODOS/STATES	Immediate and deferred

<b>RUN</b>	e.g., <b>RUN /PRODOS/DUMMY RUN</b>	<b>Immediate and deferred</b>
<b>—</b>	e.g., <b>— DUMMY</b>	<b>Immediate mode only</b>
<b>LOAD</b>	e.g., <b>LOAD /PRODOS/VIEW</b>	<b>Immediate and deferred</b>
<b>SAVE</b>	e.g., <b>SAVE /MY.DISK/DEMO</b>	<b>Immediate and deferred</b>

## QUESTIONS

1. Describe the requirements for creating pathnames.
2. Discuss the **PREFIX** command in detail. Why is it important?
3. How are files stored on a ProDOS diskette?
4. Compare the rules for naming files in ProDOS and DOS 3.3.
5. Describe the **—** (**DASH**) command. Explain why it is so handy.
6. Why is the new option to the **RUN** command so important?
7. What are the rules for volume directories and subdirectories?

# 3. HOUSEKEEPING COMMANDS

*Housekeeping in common for  
women is the acid test.*

*Andre Maurois, 1924*

## 3.0. OVERVIEW

This chapter describes those ProDOS commands that let you manipulate the files stored on your diskettes.

In general these are called housekeeping commands. This entire chapter discusses only these commands. Special attention should be given to the CREATE command and the accompanying table of file types.

In general, these are the commands that let you do things from either the immediate or the deferred modes to keep things neat, to see things, to get rid of things, or to create things.

This chapter will give you many examples of how to use each of these commands, along with the syntax required by each command.

The commands introduced in this chapter are shown in the box below.



CAT	e.g.,	CAT /PRODOS CAT /PRODOS,S6,D1	Immediate and deferred
CATALOG	e.g.,	CATALOG /PRODOS CATALOG /PRODOS,S6,D1	Immediate and deferred
PREFIX	e.g.,	PREFIX /PRODOS PREFIX /S6,D1	Immediate and deferred
CREATE	e.g.,	CREATE PIC4,TBIN CREATE DIRECTORY,TDIR	Immediate and deferred
RENAME	e.g.,	RENAME STOCKS, PORTFOLIO RENAME DEBTS,LOSSES	Immediate and deferred
DELETE	e.g.,	DELETE /MY/LOSSES DELETE /MY/DEBTS,S6,D2	Immediate and deferred
LOCK	e.g.,	LOCK /MY/PORTFOLIO	Immediate and deferred
UNLOCK	e.g.,	UNLOCK /MY/PORTFOLIO	Immediate and deferred
CHAIN	e.g.,	CHAIN PART.TWO	Immediate and deferred
STORE	e.g.,	STORE /MY/DEBTS STORE VAR.TABLE	Immediate and deferred
RESTORE	e.g.,	RESTORE /MY/DEBTS	Immediate and deferred
PR#	e.g.,	PR#6 PR#0	Immediate and deferred
IN#	e.g.,	IN#2 IN#0	Immediate and deferred

### 3.1. THE CAT AND CATALOG COMMANDS

These two commands allow you to view the names and other characteristics of the files you have stored on a diskette. By this time you should feel comfortable with both of these, since you used them in Chapters 1 and 2. The syntax of these commands has the following form:

]CAT [pn] [,S#] [,D#]
-----------------------

]CATALOG [pn] [,S#] [,D#]
---------------------------

When you use the CAT or CATALOG command you will see an entirely different screen presentation from DOS 3.3. CAT is not supported in DOS 3.3. CATALOG is the same command for both DOS 3.3 and ProDOS; however, this command produces a lot more information on the video screen, especially when you are in the 80-column format under ProDOS.

Let's look first at the CAT command. From the immediate mode type:

]CAT

You should see something like:

/PRODOS

NAME	TYPE	BLOCKS	MODIFIED
*PRODOS	SYS	31	1-OCT-83
*BASIC.SYSTEM	SYS	21	1-OCT-83
*STARTUP	BAS	1	15-JUL-83

BLOCKS FREE: XXX

BLOCKS USED: XXX

At the top of the screen is the volume name, /PRODOS in this case. The diskette in the disk drive accessed will determine the volume name presented.

The next set of data is a line-by-line presentation of the files, the files in the volume directory in this case.

The first column of data tells you if a file is locked or unlocked. If the file is locked, ProDOS uses the familiar asterisk (\*) character. If no character is present, then the file is unlocked.

The second column gives you the name of the file. Remember the file name rules?

The third column shows you the type of the file. All of the file type abbreviations are shown in the next section.

The next column gives you the number of blocks of storage space required to store the file. A block of storage is 512 bytes.

The last column gives you the last date the file was modified. If ProDOS does not know the date the term <NO DATE> will be placed in this area.

After all of the files in the directory have been listed, there is a summary line given that tells you how many blocks have been used and how many blocks are still available.

In Chapters 1 and 2 you used these commands. If you use the CATALOG command when in 40 columns, the information presented will occupy 2 lines on your video screen, because the information will wraparound on the screen. In Chapter 2 you invoked the 80-column board presentation and looked at the information presented. In order to invoke an 80-column presentation, it was only necessary to address the slot that contains the board. It is done with:

```
]PR#3
```

Once you are in the 80-column format, the CATALOG command will give you even more information than the CAT command. The only additional information that needs to be explained is the final column. This is the ENDFILE SUBTYPE column. This column gives you the number of bytes of storage space required on the diskette. For most files, this is a single number. However, in the case of a binary file, there is an address listed. This address is where the binary file will be loaded into memory, if you do not override the loading address by using the address option. This is explained in Chapter 6 when the BLOAD command is discussed.

The other possible entry in this column is an R parameter. This parameter gives you the record length of an individual record in a random-access file. Random-access files are discussed in Chapter 5.

#### *Examples of the CAT and CATALOG commands*

```
CAT
CATALOG
CAT,S6,D2
CATALOG,S6,D2
CAT /MY/NEW.DIR
CATALOG /MY/NEW.DIR
```

## 3.2. THE PREFIX COMMAND

There are times when you will be referring to a set of files within a single directory that have the same pathname except for the names of the files that contain the information you wish to retrieve. It is very tedious, cumbersome, and error prone to type the entire pathname each time you wish to retrieve one of these files. By using the PREFIX command, you can set the prefix to the name of the directory or a partial pathname. This will allow you to refer to files by their name only. This command was not available in DOS 3.3.

In order to assign a new value to the prefix or display the current prefix, use the command:

]PREFIX [pn] [,S#] [,D#]

Let's assume that your current prefix is /PRODOS/ and you want to load the program EXAMPLE located on the diskette in drive 2. The volume name of that diskette is /SCRATCH.DISK/. The entire pathname for the EXAMPLE file is /SCRATCH.DISK/EXAMPLE. In order to load that program into memory, you would have to type the entire pathname for that program. It would be much easier to set the prefix to a new partial pathname: then you will need to only refer to the file name by itself. Use the command:

```
]PREFIX /SCRATCH.DISK
```

Now, you can refer to EXAMPLE by its name only.

When you first power up your system, the PREFIX variable is left blank or empty. The slot and drive defaults are set to the slot and drive values that contain the diskette used during the powerup phase. When the prefix is empty, ProDOS uses the default slot-drive combination to find files stored on a diskette because no other information is available.

#### *Option    Description*

- |       |   |
|-------|---|
| [pn]  | pn <i>must</i> be the pathname or partial pathname of a directory file. When you assign a prefix, your system will test all peripheral storage devices looking for a valid match for the new PREFIX. If no match is found a FILE NOT FOUND or FILE TYPE MISMATCH error results. |
| [,S#] | If you specify the slot and drive instead of a file name,   |
| [,D#] | Then the volume name of the indicated diskette is assigned to PREFIX. On one drive systems, refer to that drive using both the slot number and drive 1.   |

To determine what the current PREFIX contains, type:

```
]PREFIX
```

Let's now take a couple of examples to see how this works. Assume that the ProDOS diskette is installed in disk drive 1. Set the prefix to indicate the volume directory. Examples are shown in Figure 3.1.

<i>Setting PREFIX</i>	<i>PREFIX Value</i>
]PREFIX /PRODOS	/PRODOS/
]PREFIX /PRODOS,S6,D1	/PRODOS/
]PREFIX ,S6,D1	/PRODOS/

```

]PREFIX /PRODOS/DOGS  (FILE NOT FOUND)
]PREFIX /              (blank)

```

Figure 3.1. PREFIX examples.

There are some things you need to notice. First, ProDOS adds a slash to your prefix if you do not supply the ending delimiter. Second, ProDOS will supply the volume directory when you specify the slot and drive numbers. This is very handy when you do not remember the prefix associated with a volume. Power up your system and try some of these options.

When you start up /PRODOS or any other ProDOS diskette, the value in the prefix buffer is left blank. The slot and drive defaults are set to indicate the disk drive containing that diskette. When the prefix is blank, ProDOS will look for files located on the diskette designated by the default slot and drive.

If you do not use the options with the PREFIX command, the current value of the prefix is used.

In your own programs, when you use the PREFIX command with no options, the next INPUT statement in your program expects to read the prefix. If you do specify options, then the prefix is not displayed, but it is assigned the new value.

Assume that you have ProDOS booted with the /SCRATCH.DISK disk in drive 1. You now want to determine the current prefix under program control. Let's write a program to do that.

```

]NEW
]LIST

100 D$ = CHR$(4): REM CHR$(4) = CTRL-D
110 PRINT D$;"PREFIX"
120 INPUT PF$
130 REM
140 REM * REST OF PROGRAM
150 REM

200 PRINT D$;"PREFIX ";PF$

210 END

```

<i>Lines</i>	<i>Description</i>
Line 100	Set CTRL-D to signify a ProDOS command.
Line 110	ProDOS command for PREFIX.
Line 120	Read the prefix in PF\$ variable.

Lines 130–150	Remark statements.
Line 200	Restore the prefix to the original value.
Line 210	Terminate the program.

You give the prefix command without any options, as discussed above, and then INPUT the PREFIX value into a string variable. Later in your program you may then restore the old value before leaving the program.

### 3.3. THE CREATE COMMAND

The purpose of the CREATE command is to create files of all types, although its primary purpose is to create directory files. ProDOS files can also be created using other commands. A volume directory file can store the names and locations of a maximum of 51 files. It is recommended that you create directories on a diskette before placing any other files on that diskette. DOS 3.3 does not have this command. To create a file use the syntax that follows:

```
]CREATE pn [,Ttype] [,S#] [,D#]
```

Notice that the CREATE command has a new and different option from the options shown previously in this chapter. This option is the Ttype, which determines the type of file to be created. If the type option is left out, then a directory file is created. Therefore, if you wish to create a file other than a directory, you *must* use the Ttype option.

For example, you may create a directory file named /CHECK.BOOK using the command:

```
]CREATE /CHECK.BOOK
```

The number of files you may place into a directory is limited only by the space available on a diskette. The size of a directory file is determined by the number of files in that directory. The volume directory name is the only exception to this rule. The volume directory name is created when the volume is formatted.

The first storage block (512 bytes) of diskette space used by the directory can hold a maximum of 12 file names. After that, each additional directory block can hold 13 file names.

#### *Option      Description*

pn	pn is the pathname or partial pathname of the file that you are creating. Remember the file you are creating must not already exist.
[,Ttype]	T defines the fact that the next three letters is the type designator for the file to be created. The file type abbreviations are shown in Figure 3.2. The figure shows a number of file characteristics. The only column of interest at this time is the abbreviation column.

<i>File type</i>	<i>Abbreviation</i>	<i>Values</i> <i>Hex = Dec</i>	<i>Notes</i>
Typeless file		\$00 = 0	(SOS and ProDOS)
Bad Block file	BAD	\$01 = 1	
Pascal code file	PCD	*\$02 = 2	
Pascal text file	PTX	*\$03 = 3	
ASCII text file	TXT	\$04 = 4	(SOS and ProDOS)
Pascal data file	PDA	*\$05 = 5	
Binary file	BIN	\$06 = 6	(SOS and ProDOS)
Font file	FNT	*\$07 = 7	
Graphics screen file	FOT	*\$08 = 8	
Business BASIC program	BA3	*\$09 = 9	
Business BASIC data	DA3	*\$0A = 10	
Word processor file	WPF	*\$0B = 11	
SOS system file	SOS	*\$0C = 12	
SOS reserved		*\$0D = 13	
SOS reserved		*\$0E = 14	
Directory file	DIR	\$0F = 15	(SOS and ProDOS)
RPS data file	RPD	*\$10 = 16	
RPS index file	RPI	*\$11 = 17	
SOS reserved types		*\$12 = \$BF	(reserved - SOS)
ProDOS reserved types		\$CO = \$EF	(reserved - ProDOS)
ProDOS added commands	CMD	\$F0 = 240	
ProDOS user defined	\$F#	F1-\$F8	(1-8 files)
ProDOS reserved		\$F9 = 249	
Integer BASIC program	INT	\$FA = 250	
Integer BASIC variables	IVR	\$FB = 251	
Applesoft II BASIC	BAS	\$FC = 252	
Applesoft II variables	VAR	\$FD = 253	
Relocatable code file	REL	\$FE = 254	
ProDOS system file	SYS	\$FF = 255	

## Notes:

1. The \* character designates Apple III SOS only.
2. The \* marked file types are not used by ProDOS.
3. There are a number of values reserved for specific operating systems other than ProDOS.
4. There are abbreviations you may never come across unless you have an Apple III.

Aside: (a) Extensions to ProDOS could easily be made to include additional file types.  
 (b) Notice the capability for you to add file types of your own now.

[,S#]-The slot and drive options have their normal meanings.

[,D#]

The new CREATE command gives you a very handy way to test for the existence or nonexistence of files.

*Examples for the CREATE command*

CREATE /MY/BANK	DIR file created
CREATE /MY/BANK,TDIR	DIR file created
CREATE /MY/BANK/NOTES,TDIR,S6,D2	DIR file created
CREATE /MY/CREDIT.CARDS,TTXT	TXT file created
CREATE MY/TIME.SYSTEM,TSYS,D1	SYS file created

### 3.4. THE RENAME COMMAND

The RENAME command is for changing the name of a file that is already stored on a diskette. This command operates essentially the same way as the RENAME command in DOS 3.3. The syntax of the command is:

```
]RENAME pn1,pn2 [,S#] [,D#]
```

The pn1 is the pathname of a file that is to be changed to pn2. The only caution is that pn2 *must* be in the same directory as pn1. Therefore you can use:

```
]RENAME /MY/STOCKS/PERSONAL,/MY/STOCKS/PORTFOLIO
```

to change the PERSONAL file's name to PORTFOLIO. However, you *cannot* use the command:

```
RENAME /MY/PERSONAL/STOCKS,/MY/STOCKS/PORTFOLIO
```

If you need to move a file from one directory to another, it is usually necessary to use the FILER capability. This is discussed in Chapter 8.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn1,pn2	pn1 and pn2 are the pathnames that indicate the storage location of the file. Both of these pathnames must be unique. There are a number of possible errors that could result from using incorrect pathnames.
[,S#]	Both the slot and drive options have their usual meanings.
[,D#]	

*Examples for the RENAME command*



```

RENAME /MY/DEBTS/OWED,/MY/DEBTS/PAID
RENAME /MY/STOCKS/OWNED,/MY/STOCKS/SOLD
RENAME GAME.PROGRAM,WORD.GAME
RENAME WORD.PUZZLE,CRYPTOGRAM

```

### 3.5. THE DELETE COMMAND

The purpose of this command is to remove a file from the diskette. This command operates exactly the same as in DOS 3.3. The syntax of the instruction is:

]DELETE pn [,S#] [,D#]
------------------------

You can remove the file /MY/RESUME from a diskette with the command:

```
]DELETE /MY/RESUME
```

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	A pathname or partial pathname. The pathname must be included and the file must exist on the diskette before you may use this command.
[,S#]	Both the slot and drive options have their usual meanings.
[,D#]	

*Examples for the DELETE command*

```

DELETE /MY/DEBTS/OWED
DELETE /MY/BANK/NOTES/PAID
DELETE WORD.GAME
DELETE MATH.DEMO

```

### 3.6. THE LOCK AND UNLOCK COMMANDS

There are times when you will want to protect your files from accidentally being changed, deleted, or renamed. This can be done using the LOCK command. Then in order to make a revision to the locked file, you will be required to use the UNLOCK command. Both of these commands operate exactly the same as in DOS 3.3. The syntax of these commands is:

]LOCK pn [,S#] [,D#]
----------------------

]UNLOCK pn [,S#] [,D#]
------------------------

For example, to lock a file use:

```
]LOCK /MY/XMAS.LIST
```

and later to unlock that same file use:

```
]UNLOCK /MY/XMAS.LIST
```

As long as a file is locked, you cannot rename, delete, or change that file in any way without first unlocking that file.

For example, let us assume you are going to run both parts of a program.

```
]RUN /MY/FIRST.PART
```

When you are finished with this program, you need to execute the following:

```
]CHAIN /MY/SECOND.PART,@1000
```

Notice that the program, SECOND.PART, commences execution at line 1000. This could be for a number of reasons. The most common might be because any chained program may not dimension any array used in any previous part of a program.

#### *Option    Description*

pn	pn is the pathname or partial pathname that contains the Applesoft II BASIC program needed to be run next.
[, @#]	This option, when used, specifies the line number at which program execution is to begin. If the specified line number does not exist, the next higher line number in the program begins the execution. If this option is not used, program execution begins with the lowest line number.
[, S#]	Both the slot and drive options have their usual meanings.
[, D#]	

If you look at a catalog of the files on a directory or volume, the locked files will be shown with an asterisk preceeding the file information presented on the screen. This is the same as in DOS 3.3.

#### *Option    Description*

pn	pn is the pathname or partial pathname of the file that is to be either locked or unlocked. Note: You cannot lock a volume name.
[, S#]	Both the slot and drive options have their usual meanings.
[, D#]	

*Examples for the LOCK and UNLOCK commands*

```

]UNLOCK /MY/BANK/NOTES/PAID
]LOCK WORD.GAME
]UNLOCK WORD.GAME
]LOCK /MY/BANK/NOTES/PAID

```

**3.7. I/O FROM PROGRAMS**

In this section the commands covered are those that enable you to communicate easily with other Applesoft II BASIC programs and peripheral devices such as printers, disk drives, MODEMS, and clocks. When the Apple computers communicate with peripheral devices, this is known as either the input or output of information. Generically, this is referred to as input/output or just I/O.

**3.7.1. The CHAIN command**

The CHAIN command is for executing separate parts of a program in sequential order. This is very handy when a program becomes very large and will not all fit into memory. Now, you have the capability to develop your programs in parts and then chain them at execution time. Assume that you have a two-part program. You run the first part. Now, you need to run the second part. The variables and files open from the first part will be preserved when you chain the second part to the first part.

DOS 3.3 does not support this command in conjunction with Applesoft II BASIC. However, there was a binary program on the SYSTEM MASTER diskette called CHAIN that performed this function. This has now been incorporated into ProDOS. The syntax for the CHAIN command is:

```
]CHAIN pn [,@#] [,S#] [,D#]
```

*Examples for the CHAIN command*

```

PRINT DS;"CHAIN SECOND.PART"
PRINT DS;"CHAIN SECOND.PART,@1000"
PRINT DS;"CHAIN FIRST.PART"
PRINT DS;"CHAIN FIRST.PART,@2000,S6,D1"

```

**3.7.2. The STORE command**

This command and its counterpart, RESTORE, are really one of the more exciting additions made to this new operating system. This command allows you to store the names and values

### 3.7.3. The **RESTORE** command

This command allows you to bring previously STOREd names and variable values into your current program. Only a file previously STOREd can be RESTOREd. This command is not supported in DOS 3.3. The syntax of the command is:

]STORE pn [,S#] [,D#]

of all of the variables active in an Applesoft II BASIC program. This command is not available in DOS 3.3.

This is very handy if you wish to save the condition of a game during play, the condition of long numerical methods calculations, or your position in a long mailing list. The syntax of this command is:

]RESTORE pn [,S#] [,D#]

The STORE command places the variables in a file of the VAR type. When you execute the STORE command, it may take some time before the disk drive comes on and actually stores the variables. The reason for this is because ProDOS is compacting the information before storing the data.

#### *Option    Description*

pn	pn is the pathname or partial pathname of the file in which the variables are to be stored.
[,S#]	Both the slot and drive options have their usual meanings.
[,D#]	

#### *Examples for the STORE command*

```
PRINT DS;"STORE EXAMPLE.DATA"
PRINT DS;"STORE /MY/XMAS.LIST.DATA,S6,D1"
```

When you use this command, all currently defined variables are cleared from memory before the new ones are brought into memory.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn is the pathname or partial pathname of the file containing the program variables.
[,S#]	Both the slot and drive options have their usual meanings.
[,D#]	

*Examples for the RESTORE command*

```
PRINT D$;"RESTORE EXAMPLE.DATA"
PRINT D$;"RESTORE /MY/XMAS.LIST.DATA,S6,D1"
```

**3.7.4. The PR# command**

This command is used to transfer data and information from the normal video screen output to the device connected to the peripheral slot specified in the command. This command, PR#, is used to transfer the destination of such data or information. This command is the same as in DOS 3.3. The syntax for this command is:

]PR# slot

The slot specified must be in the range from 0 through 7 on the Apple II Plus and in the range from 1 through 7 on the Apple IIe.

For example, assume that your systems printer interface card is installed in slot 1. Then the command:

```
]PR# 1
```

causes all subsequent data and information to be sent to the printer. When you wish to return data output to the video screen, it is only necessary to issue the correct command. This command is:

```
]PR# 0
```

If you have an 80-column card installed in slot 3, then you may activate an 80-column screen presentation by issuing:

```
]PR#3
```

from the immediate mode. Your video display will then change to the 80-column format. On the Apple IIe the Apple Computer 80-column card or extended 80-column card is installed in

the auxiliary slot. The auxiliary slot is mapped into slot 3 for purposes of activating and deactivating an 80-column card. On an Apple II Plus, an 80-column card is normally installed in slot 3.

*Examples for the PR# command*

```
PRINT D$;"PR# 1"
]PR#1
PRINT D$;"PR# 2"
]PR#2
PRINT D$;"PR# 0"
]PR#0
```

### 3.7.5. The IN# command

Normally, Apple computers will receive data and information from the keyboard. The IN# allows you to receive data and information from sources other than the keyboard. This command works the same as in DOS 3.3. The syntax of the command is:

]IN# slot
-----------

For example, assume that you have a MODEM in slot 2 and that you are going to receive data from your friend out of state. Your software would use the command:

```
]IN# 2
```

When you are through receiving data, you can return your system to the normal or default condition by using the command:

```
]IN# 0
```

The comments that apply to an 80-column card for the PR# command also apply to the IN# command.

If you are going to both receive and transmit data through one particular slot, you will need to issue both the IN# and PR# commands.

*Examples for the IN# command*

```
PRINT D$;"IN# 2"
]IN#2
```

```
PRINT D$;"IN# 0"
JIN#0
```

## SUMMARY

This chapter discussed housekeeping commands that are supported in ProDOS.

The main thrust of this chapter was to introduce you to the commands that you will use frequently to manipulate or change the status of files.

Special emphasis was given to the CREATE command and all of the currently possible file types.

Two commands that deal with peripheral devices were also shown. These two were the IN# and PR# commands. It is through the use of these commands that Apple II computers are able to communicate with the outside world by using ProDOS or Applesoft II BASIC. There are other ways of course, but they are not a part of the discussion here.

A number of examples of each command were given along with the rules for forming the commands.

The commands introduced were:

CAT	e.g.,	CAT /PRODOS CAT /PRODOS,S6,D1	Immediate and deferred
CATALOG	e.g.,	CATALOG /PRODOS CATALOG /PRODOS,S6,D1	Immediate and deferred
PREFIX	e.g.,	PREFIX /PRODOS PREFIX /S6,D1	Immediate and deferred
CREATE	e.g.,	CREATE PIC4,TBIN CREATE DIRECTORY,TDIR	Immediate and deferred
RENAME	e.g.,	RENAME STOCKS,PORTFOLIO	Immediate and deferred
DELETE	e.g.,	DELETE /MY/LOSSES DELETE /MY/DEBTS,S6,D2	Immediate and deferred
LOCK	e.g.,	LOCK /MY/PORTFOLIO	Immediate and deferred
UNLOCK	e.g.,	UNLOCK /MY/PORTFOLIO	Immediate and deferred
CHAIN	e.g.,	CHAIN PART.TWO	Immediate and deferred
STORE	e.g.,	STORE /MY/DEBTS	Immediate and deferred
RESTORE	e.g.,	RESTORE /MY/DEBTS	Immediate and deferred
PR#	e.g.,	PR#6 PR#0	Immediate and deferred
IN#	e.g.,	IN#2 IN#0	Immediate and deferred

## QUESTIONS

1. How do you invoke the 80-column card?
2. What is the function of the CREATE command? Explain in detail.
3. How do you retrieve, save, and later restore the current PREFIX?
4. How do you use the PR# and IN# commands? How can you communicate with the outside world using these commands?
5. What is the difference between the CAT and CATALOG commands?



# 4. SEQUENTIAL- ACCESS FILES

*Have an open face, but  
conceal your thoughts.*

*Italian Proverb*

## 4.0. OVERVIEW

This chapter gives you your first experience with text files. You will be shown how to create, store data into, and retrieve data from sequential text files.

In the early part of this chapter a description of a sequential-access file is given. Throughout this chapter, programs and program segments will be given illustrating the commands being described. The last sections of this chapter describe the commands needed to operate with sequential text files.

There are many times that you may wish to store information or data that is not a program onto a diskette. You may wish to keep track of your stock portfolio, municipal bonds, recipes, or addresses. ProDOS allows you to do just that.

To do this, create a program that establishes a file on a diskette. That file will contain the information you wish to keep. Next, create a program that will request information from the computer user and store the data on the diskette in the file. Last, it would be nice to have a program to recall the stored data and display it on the screen or print it on paper in a printer.

The three possible programs discussed may all be contained within one program, as major subroutine modules. In fact there are only about seven major evolutions that need to be performed on files. These are shown in Figure 4.1.

```

Creation program
  Create file
  Delete file

Processing program
  Enter data
  Change data
  Delete data

Output program
  Review file data
  Report printing

```

Figure 4.1. File programs.

A file is like a list of items stored in some order, similar to a telephone directory or a shopping list. This list may be of any size, up to the capacity of a diskette, and contain any information or data—even a program. Files may be created under the control of ProDOS using the TEXT (TXT) type files. The reason they are called TEXT files is that data is stored in the file in essentially text form. ProDOS supports two types of TEXT files:

```

—sequential-access
—random-access

```

There is no essential difference in the form of data storage, only in the file characteristics. Random-access files are covered in the next chapter.

The letters TXT in the third column of a CAT display denotes a text file.

Since text files are NOT programs, but data, you cannot RUN, SAVE, or LOAD them. Other commands, covered later, may be used with text files.

The commands introduced in this chapter are shown in the box below.

OPEN	e.g.,	OPEN /MY/XMAS.LIST	Deferred mode only
READ	e.g.,	READ /MY/XMAS.LIST	Deferred mode only
WRITE	e.g.,	WRITE /MY/XMAS.LIST	Deferred mode only
CLOSE	e.g.,	CLOSE CLOSE /MY/XMAS.LIST	Immediate and deferred
APPEND	e.g.,	APPEND /MY/XMAS.LIST	Deferred mode only

FLUSH	e.g.,	FLUSH /MY/XMAS.LIST	Immediate and deferred
POSITION	e.g.,	POSITION /MY/XMAS.LIST,F3	Deferred mode only

## 4.1. SEQUENTIAL-ACCESS FILES

The sequential-access file is a linear list of items, words, or numbers in which each item is considered a record in the file. This type of file organization comes from the days when the only auxiliary storage media available to computers were high-speed tape drives. Tape drives require you to start at the beginning of a file, read each record in sequence, and perform any processing sequentially. The sequential-access file stored on a diskette has the same characteristics. When you either READ or WRITE data to the diskette, it is done in sequential order.

To create the file, store data into the file on the diskette, and retrieve data from the diskette, you must first OPEN communications between the program and ProDOS. With communications established with ProDOS, you next have to tell ProDOS the type of communications required; whether you wish to READ or WRITE data. Once an Applesoft II BASIC program has informed ProDOS that you wish to communicate and the type of communication desired, you may either INPUT or PRINT data respectively. Once communication is finished, you must CLOSE the communications conversation with ProDOS.

The text file commands have a number of options. A few of them you will use most of the time. Some of the options you will use only rarely. As each command is described, examples will be given that will illustrate the command and the option.

The basic unit of a sequential-access text file is the field. A field is a series of characters that has as its final character the carriage return (RETURN) character. In Applesoft II BASIC you have three options when printing a line of text. These are shown in the small program below.

The rest of this section will describe and explain a number of program examples using the instructions introduced in the chapter. After the instructions have been used, the remainder of the chapter gives you the syntax and additional information about the instructions that have been used.

```
]NEW
```

```
]LIST
```

```
]100 A$ = "APPLE": B = "SOFT"
]110 PRINT A$
]120 PRINT A$;B$
]130 PRINT A$,B$
```

```
]RUN
```

```
APPLE
APPLESOFT
APPLE    SOFT
```

After you have entered this small program, RUN it and you will see the three lines of text printed on the screen as shown above. Notice that there is a carriage return character at the end of each line. It is that character that causes the cursor to return to the left margin on the next sequential display line. The delimiters (semicolon and comma) between variables in the print statements do not cause the termination of a line of printed text.

This same scheme is used when you print data to a text file. When you print data to a sequential-access file using Applesoft II BASIC print statements without a semicolon or comma, the field is terminated with a carriage return.

A carriage return character signifies the end of a field. The next print statement will store data into the next field of the file. Each subsequent print statement ending in a carriage return stores data into the next field in the file. In this way a sequential-access text file may contain any number of fields.

Let's now look at how data is stored into a sequential-access file. Assume that you have the program segment:

```
]NEW

]LIST

10  REM* NUMBERS PROGRAM
30 D$ = CHR$(4)
   :

100 PRINT D$;"OPEN NUMBERS"
110 PRINT D$;"WRITE NUMBERS"
120 PRINT "ONE"
130 PRINT "TWO"
140 PRINT "THREE"
150 PRINT "FOUR"
160 PRINT D$;"CLOSE NUMBERS"
500 END
```

It is recommended that you power up your system with the /SCRATCH.DISK in the boot drive. When you have the Applesoft II BASIC prompt character and cursor you are ready to enter the above program. By the way, you will add to this program later.

<i>Lines</i>	<i>Description</i>
--------------	--------------------

Line 10	Remark statement identifying program.
---------	---------------------------------------

Line 30    The variable D\$ contains the CTRL-D ProDOS character that signifies the following instruction is a ProDOS command.  
 Line 100   OPEN communications with ProDOS. OPEN is discussed in Section 4.2.  
 Line 110   The communications is to WRITE data to the diskette.  
 Line 120   Store the first field to the diskette.  
 Line 130   Store the second field to the diskette.  
 Line 140   Store the third field to the diskette.  
 Line 150   Store the fourth field to the diskette.

*Lines 120 through 150 do not have the CTRL-D preceding the operation. This is because all output has already been redirected to the file through the instruction in line 110.*

Line 160   Terminate communications with ProDOS. CLOSE is discussed in Section 4.5.  
 Line 500   END the program.

When you have this program entered; SAVE it as /SCRATCH.DISK/NUMBER.PROGRAM and then RUN it. The information you have stored on a diskette is stored in the following way under the name NUMBERS. This shown in Figure 4.2.

This particular file has four fields that encompass 19 characters, including all of the carriage returns. Please notice that the first field in the file is designated 0.

The next obvious question that arises is "How can I see what has been stored in the file?" Fair question. In the next few paragraphs more will be added to the program /SCRATCH.DISK/NUMBER.PROGRAM.

Load the program /SCRATCH.DISK/NUMBER.PROGRAM from your disk driver and LIST the program. Then enter the following additional code:

```
20 DIM A$(10)
   :
200 PRINT D$;"OPEN NUMBERS"
210 PRINT D$;"READ NUMBERS"
220 INPUT A$(1)
230 INPUT A$(2)
240 INPUT A$(3)
```

Characters stored:	ONE	TWO	THREE	FOUR
Field numbers:	{0}	{1}	{2}	{3}

Notes: (1) The | character represents a carriage return.  
 (2) Numbers represent the field numbers.

Figure 4.2. Sequential file character storage.

```

250 INPUT A$(4)
260 PRINT D$;"CLOSE NUMBERS"
300 FOR I = 1 TO 4: PRINT A$(I): NEXT I

```

<i>Lines</i>	<i>Description</i>
--------------	--------------------

Line 20	Dimension the variable A\$ to handle 11 elements.
Line 200	OPEN communications with ProDOS. OPEN is discussed in Section 4.2.
Line 210	The communications is to READ data from the diskette. READ is discussed in Section 4.3.
Line 220	Read the first field from the diskette.
Line 230	Read the second field from the diskette.
Line 240	Read the third field from the diskette.
Line 250	Read the fourth field from the diskette.

*Lines 220 through 250 do not have the CTRL-D preceding the operation. This is because all input has already been redirected to the file through the instruction in line 210.*

Line 260	Terminate communications with ProDOS. CLOSE is discussed in Section 4.5.
Line 300	Print the contents of the A\$ vector to the video screen.

When you have the additional program code entered, SAVE it as /SCRATCH.DISK/NUMBER.PROGRAM and then RUN it again. By the way, you will add more to this program later. If everything goes well, you should see displayed on the screen:

```

ONE
TWO
THREE
FOUR

```

```

]
```

You now know that what you stored may be retrieved and seen through the mechanism of supporting code. Also, you have now written a very simple data storage and retrieval system. It really wasn't very difficult, was it?

Earlier in this section, three ways to print information to a screen were described. In Figure 4.3 the same options will be shown for printing to a sequential text file. The variables A\$ and B\$ are the same as before. A\$ = "APPLE" and B\$ = "SOFT". The | character still represents a carriage return character.

<i>Statement</i>	<i>Characters</i>	<i>Comments</i>
PRINT "APPLE"	APPLE	Complete field.
PRINT A\$;	APPLE	Partially completed field.

PRINT B\$	APPLESOFT	Completed field. Notice that there is no carriage return after the A\$ characters in the file.
PRINT A\$,B\$	APPLESOFT	Completed field. Notice that there are no spaces placed in the field.
PRINT B\$;" ";A\$;"S"	SOFT APPLES	Completed field. Notice how you can link items in a file field.

Figure 4.3. Print to a text file.

The reason for showing you these options is to give you a feel for how to put characters together to form fields in a sequential-access text file.

Each field is written or read from a sequential-access text file through the use of a single variable. The variables shown, thus far, have been string variables. You may also use numeric variables, either floating point or integer. Each of these may be interspersed with string variables. The only caution is that you must know the order in which you stored data so that it can be retrieved correctly.

Figure 4.4 shows you ways in which you may retrieve the data stored in a file.

<i>Statement</i>	<i>Characters</i>	<i>Comments</i>
INPUT A\$	APPLE	Reads one complete field.
INPUT B\$	SOFT	Reads one complete field from next field in the file.
INPUT A\$,B\$	APPLE SOFT	Reads two complete adjacent fields from file.
GET C\$	A	Reads a single character from field in a file. If you are going to use the GET statement, you will have to write a loop in code that: <ol style="list-style-type: none"> <li>1. Reads and links character read to some string variable.</li> <li>2. Tests each character read for the field delimiter (ASCII 13).</li> </ol>

Figure 4.4. Retrieving from a text file.

There are several ways to retrieve data from a sequential-access text file. In most cases, the INPUT statement is the best. However, there are times when the GET statement with supporting code is required.

Up to this point, there has been one field in each record of a sequential-access text file. There is a way to have multiple elements of data in one field. By this I mean that there is a way to place multiple pieces of data within a single field of a sequential-access file. It may be done by physically placing commas between each element that makes up each field in the file. For example, let's put three elements of information within one field using the following code:

JNEW

JLIST

```

10  REM* MULTI-ELEMENTS PROGRAM
20  DIM A$(10)
30  D$ = CHR$(4)
    :

100 PRINT D$;"OPEN ELEMENTS.DATA"
110 HOME: PRINT TAB( 10) "MULTI-ELEMENTS DEMO": PRINT
120 PRINT D$;"WRITE ELEMENTS.DATA"
130 PRINT "FIRST,SECOND,THIRD"
140 PRINT "ELEMENTS,PER,FIELD"
150 PRINT D$;"CLOSE"
200 PRINT D$;"OPEN ELEMENTS.DATA"
210 PRINT D$;"READ ELEMENTS.DATA"
220 FOR I = 1 TO 3
230 INPUT A$(I): NEXT I
240 FOR I = 4 TO 6
250 INPUT A$(I): NEXT I
260 PRINT D$;"CLOSE"
300 FOR I = 1 TO 6 : VTAB 2+I
310 PRINT A$(I)
320 NEXT I
500 END

```

Power up your system with the /SCRATCH.DISK in the boot drive. Then enter this program from the keyboard. When it has been entered, save the program using the name MULTI.ELEMENTS. Then RUN the program to see what happens.

You should get the following output:

MULTI-ELEMENTS DEMO

FIRST  
SECOND  
THIRD  
ELEMENTS  
PER  
FIELD

]



<i>Lines</i>	<i>Description</i>
Line 10	Remark statement identifying program.
Line 20	Dimension the variable A\$ to handle 11 elements.
Line 30	The variable D\$ contains the CTRL-D ProDOS character that signifies the following instruction is a ProDOS command.
Line 100	OPEN communications with ProDOS. OPEN is discussed in Section 4.2.
Line 110	Clears screen. HOME. Prints screen title centered at the top. Print a blank line on the screen. PRINT.

*It is permissible to print to the screen after a file is open provided you do the printing before the WRITE instruction to the file is executed.*

Line 120	The communications is to WRITE data to the diskette.
Line 130	Store the first field to the diskette. This field contains three elements separated by commas within the quoted field string.
Line 140	Store the second field to the diskette. This field also contains three elements separated by commas within the quoted string.

*Lines 130 and 140 do not have the CTRL-D preceding the operation. This is because all output has already been redirected to the file through the instruction in line 120.*

Line 150	Terminate communications with ProDOS. CLOSE is discussed in Section 4.5.
Line 200	OPEN communications with ProDOS. OPEN is discussed in Section 4.2.
Line 210	The communications is to READ data from the diskette. READ is discussed in Section 4.3.
Line 220	Top of FOR—NEXT I loop.
Line 230	Read the first field from the diskette. This field contains three elements. Range of the I loop. NEXT I.
Line 240	Top of FOR—NEXT I loop.
Line 250	Read the second field from the diskette. This field also contains three elements. Range of the I loop. NEXT I.

*Lines 230 and 250 do not have the CTRL-D preceding the operation. This is because all input has already been redirected to the file through the instruction in line 210.*

Line 260	Terminate communications with ProDOS. CLOSE is discussed in Section 4.5.
Line 300	Top of the FOR—NEXT I loop. Position cursor on the screen. VTAB 2+I.

Line 310    Print the contents of the A\$ vector to the video screen.  
 Line 320    Range of the I loop. NEXT I.  
 Line 500    END the program.

Notice that lines 130 and 140 from the program are actually written to the file with commas embedded within the quoted string. Later in the program, you will need to retrieve the six pieces of data. Since INPUT statements consider commas as delimiters signifying the end of an element, it is necessary to use multiple inputs to read an entire field. This has been shown in lines 220 through 250.

Line 230 could be replaced with:

```
230 INPUT A$(1),A$(2),A$(3)
```

and line 250 could have been written:

```
250 INPUT A$(4),A$(5),A$(6)
```

If you choose to write these lines this way, then the FOR—NEXT loop code will need to be deleted.

At this point, it might be a good idea to look at how these two multiple element fields are stored on a diskette. This is shown in Figure 4.5.

Now, using the information presented, how would you read only the first element of each field? Another question could be: how could you use the GET instruction to retrieve information from the ELEMENTS.DATA file? I am sure you can think of many other possible iterations and questions.

## 4.2. THE OPEN COMMAND

The OPEN command is required as the preparatory command that tells ProDOS to prepare for the reading or writing of data and information from a diskette. It also prepares a file buffer space in memory to hold data. This is the command that tells ProDOS you wish to communicate. This command operates essentially the same as in DOS 3.3. The syntax for this command is:

]OPEN pn [,S#] [,D#]

After you have opened communication with a file, you are allowed to either read or write data from or to that file. Remember that any open file should be closed before you end the current program that is executing.

Characters stored:	FIRST,SECOND,THIRD	ELEMENTS,PER,FIELD
Field numbers:	{      0      }	{      1      }

Notes: (1) the | character represents a carriage return.  
 (2) Numbers represent the field numbers.

Figure 4.5. Sequential file character storage.

When a program opens a text file, a number of possible things take place. These are:

1. ProDOS will set up a file buffer space in memory.
2. Resets HIMEM just below the file buffer space.
3. Prepares your system to either read or write data starting at the beginning of the file. Allows for eight files to be open simultaneously.
4. If the file with the pn does not exist, that file is then created and added to the appropriate directory.
5. If the file already exists and is open, a FILE BUSY error will result when the file is reOPENed.

#### *Option      Description*

pn	pn is the pathname or partial pathname of the file that is to be opened. If the file does not already exist, a file of the text (TXT) type is created.
[,S#]	The slot and drive options have their normal meanings.
[,D#]	

#### *Examples of the OPEN command*

```
PRINT D$;"OPEN EXAMPLE"
PRINT D$;"OPEN DEMO.DAT,S6,D1"
PRINT D$;"OPEN /SCRATCH.DISK/DEMO.DAT"
PRINT D$;"OPEN /SCRATCH.DISK/DEMO.DAT,S6,D1"
```

When working with sequential-access files, one of the minor but very aggravating problems is the possibility of old information staying in the file as the total size of the file changes. This can happen because a sequential-access file has the characteristic of expanding and contracting as data within the file changes. When the file contracts, the old data is not erased as new data is placed in the file. So, in order to prevent the retrieval of the unwanted data from the old version of the file, it is recommended that you first delete the file before writing the new information to the file. This is done in the following way:

```
:
100 PRINT D$;"OPEN EXAMPLE"
105 PRINT D$;"CLOSE EXAMPLE"
```

```

110 PRINT D$;"DELETE EXAMPLE"
115 PRINT D$;"OPEN EXAMPLE"
    ;

```

This code opens the **EXAMPLE** file and then closes the file. This assures that the file exists on the diskette. Then delete the file. Immediately thereafter, the file **EXAMPLE** is opened again. This reestablishes the file on the diskette. One caution when using this code: make sure that you have everything from the old file either in memory or stored elsewhere before you delete the file. In that way, you will not lose any information.

### 4.3. THE READ COMMAND

After a file has been **OPENed**, you must further define to **ProDOS** the type of communication required; **READ** in this case. The **READ** command identifies to **ProDOS** the file and the position in the file from which characters are to be read and prepares **ProDOS** for the **INPUT** statement(s) that are to follow. Once the **READ** command has been executed, it remains in effect until the next **ProDOS** command is executed. This command operates the same as the **READ** command in **DOS 3.3**, except for the added field number (**F#**) option. The syntax for this command is:

```
]READ pn [,F#] [,B#]
```

Every time you use the **READ** command, you *must* identify the file to be read by using the **pn**.

<i>Option</i>	<i>Description</i>
---------------	--------------------

<b>pn</b>	<b>pn</b> is the pathname or partial pathname of the file that is to be opened. This must be identical to the <b>pn</b> used when you opened the file.
<b>[,F#]</b>	<b>#</b> is an integer number that indicates the number of fields past the current position that should be read and discarded. This is done by reading characters, starting from the current position, until the specified number of carriage returns has been read.

*A carriage return character is used to separate fields in a sequential-access file.*

<b>[,B#]</b>	<b>#</b> is an integer number that indicates the number of bytes, or characters, to be read and discarded. This option will change the current position in the file.
--------------	--

*Examples of the READ command*

```
PRINT D$;"READ EXAMPLES"
PRINT D$;"READ DEMO.DATA,F4"
PRINT D$;"READ /SCRATCH.DISK/NUMBERS"
PRINT D$;"READ /SCRATCH.DISK/NUMBERS,F2,B3"
```

**4.4. THE WRITE COMMAND**

After a file has been OPENed, you must further define to ProDOS the type of communication required; WRITE in this case. The WRITE command identifies to ProDOS the file and the position in the file to which characters are to be written and prepares ProDOS for the PRINT statement(s) that are to follow. Once the WRITE command has been executed, it remains in effect until the next ProDOS command is executed. This command operates the same as the WRITE command in DOS 3.3, except for the added field number (F#) option. The syntax for this command is:

**]WRITE pn [,F#] [,B#]**

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn is the pathname or partial pathname of the file that is to be written. It must be identical to the pn for the file you opened.
[,F#]	# is an integer number that indicates the number of fields past the current position that should be read and discarded. This is done by reading characters, starting from the current position, until the specified number of carriage returns has been read.

*A carriage return character is used to separate fields in a sequential-access file.*

[,B#]	# is an integer number that indicates the number of bytes, or characters, to be read and discarded. This option will change the current position in the file.
-------	---

*Examples of the WRITE command*

```
PRINT D$;"WRITE EXAMPLES"
PRINT D$;"WRITE DEMO.DATA,F2,.B3"
PRINT D$;"WRITE /SCRATCH.DISK/NUMBERS,F2"
PRINT D$;"WRITE /SCRATCH.DISK.DEMO.DATA,F3,B2"
```

## 4.5. THE CLOSE COMMAND

When you are finished reading or writing to a file, you must close the file. It is necessary to close the file in order to make sure that all characters are written to the file, the file buffer memory is released to usable memory, and unwanted errors and error messages do not occur. The CLOSE command works the same as in DOS 3.3. The syntax for the command is:

]CLOSE [pn]

If you use the CLOSE command without any options, all OPEN files will be closed and all file buffers will be released.

<i>Option</i>	<i>Description</i>
---------------	--------------------

[pn]	pn is the pathname or partial pathname that indicates the file that is to be closed.
------	--

### *Examples of the CLOSE command*

```
PRINT D$;"CLOSE"
PRINT D$;"CLOSE NUMBERS"
PRINT D$;"CLOSE /SCRATCH.DISK/NUMBERS"
PRINT D$;"CLOSE /SCRATCH.DISK/DEMO.DATA"
```

When you are developing programs of your own, an error might occur that requires you to change your code. If you make a mistake when a file or files were open, you should issue a CLOSE without options before correcting and rerunning your program.

## 4.6. THE APPEND COMMAND

This command allows you to add information to the end of a sequential-access file. This command actually is three commands in one. It opens the file, positions to the end of the file, and then performs a write to the file. This command operates the same as in DOS 3.3. The syntax of this command is:

]APPEND pn [,S#] [,D#]

The APPEND command performs the OPEN, POSITION, and WRITE functions. It performs the three commands in the one statement. Once you have coded the APPEND com-

mand, you must also code a PRINT command that will actually store the data at the end of the file.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn is the pathname or partial pathname of the file that is to be opened and written. If the file doesn't already exist, the file is created.
[,S#]	The slot and drive options have their normal meanings.
[,D#]	

*Examples of the APPEND command*

```
PRINT D$;"APPEND EXAMPLES"
PRINT A$
```

```
PRINT D$;"APPEND /SCRATCH.DISK/DEMO.DATA,S6,D2"
PRINT A$,B$
```

## 4.7. THE FLUSH COMMAND

When writing data to a text file, ProDOS stores up a block of information (512 bytes, or characters) before anything is placed on the diskette. The FLUSH command is for the purpose of making sure that any characters left in the file buffer area are stored onto the diskette. In this way you are assured that all characters destined to be written to a file have been stored in that file. This command is not supported in DOS 3.3. The syntax for this command is:

]FLUSH [pn]

If you use the FLUSH command without any options, all open files will be flushed of any leftover data, the same as with the CLOSE command.

<i>Option</i>	<i>Description</i>
---------------	--------------------

[pn]	pn is the pathname or partial pathname that indicates the file that was opened. The pn must be identical to the pn used to open the file.
------	---

*Examples of the FLUSH command*

```
PRINT D$;"FLUSH"
PRINT D$;"FLUSH EXAMPLES"
PRINT D$;"FLUSH /SCRATCH.DISK/DEMO.DATA"
```

When you use this command, remember it takes more than a normal amount of time and will have a tendency to slow down your processing. Therefore, you will have to weigh the tradeoff between data integrity and speed.

## 4.8. THE POSITION COMMAND

With this command, you are able to access the information in any field or byte within the file. This command operates similar to the same command in DOS 3.3. The syntax for this command is:

**POSITION pn,F#**

Notice that neither of the arguments is optional. This command starts at the current position in the file and then reads and discards the number of fields specified in the F# argument. Remember that the file specified in pn must first be open before you may use the POSITION command.

<i>Option</i>	<i>Description</i>
---------------	--------------------

<b>pn</b>	pn is the pathname or partial pathname that indicates the file that was opened. It is the pn of the file whose position is to be altered. It must be identical to the pn of the file opened.
<b>F#</b>	# indicates the number of fields to be read and discarded. If you try to position beyond the end of the file, you will get an END OF DATA error message.

### *Examples of the POSITION command*

```
PRINT D$;"OPEN EXAMPLES"
PRINT D$;"POSITION EXAMPLES,F3"
```

```
PRINT D$;"OPEN /SCRATCH.DISK/DEMO.DATA"
PRINT D$;"POSITION /SCRATCH.DISK/DEMO.DISK,F4"
```

## SUMMARY

This chapter introduced you to ProDOS file processing. This chapter covered the first of the two types of files supported by ProDOS. These file types are:

- Sequential-access text file
- Random-access text file



The random-access text file will be covered in Chapter 5. This chapter covered the sequential-access file.

You were given various ways of both reading and writing data to a sequential-access text file. A number of small programs were created to illustrate how to work with this type of text file.

The syntax was given for each of the commands introduced in this chapter. Examples were also given for each of the commands.

The similarities and differences between these ProDOS and DOS 3.3 commands were shown.

Remember that a sequential-access text file is very efficient in the use of diskette storage space; however, it is not as easy to use sequential-access text files as it is to use random-access text files. You will find that some of your files will better fit the sequential-access type while others will better fit the random-access type.

The commands introduced in this chapter were:

OPEN	e.g.,	OPEN /MY/XMAS.LIST	Deferred mode only
READ	e.g.,	READ /MY/XMAS.LIST	Deferred mode only
WRITE	e.g.,	WRITE /MY/XMAS.LIST	Deferred mode only
CLOSE	e.g.,	CLOSE	Immediate and deferred
		CLOSE /MY/XMAS.LIST	
APPEND	e.g.,	APPEND /MY/XMAS.LIST	Deferred mode only
FLUSH	e.g.,	FLUSH /MY/XMAS.LIST	Immediate and deferred
POSITION	e.g.,	POSITION /MY/XMAS.LIST,F3	Deferred mode only

## QUESTIONS

1. Describe how data is stored on a diskette in a sequential-access file.
2. Describe each of the commands required to write data to a sequential-access file.
3. Describe each of the commands required to read data from a sequential-access file.
4. Which command ensures that you have stored all of the data to a file? What tradeoffs are to be considered?
5. Describe how to store multiple pieces of data in one field of a sequential-access text file.
6. Explain how to use the Applesoft II BASIC GET instruction to retrieve data and information from a sequential-access text file.

# 5. RANDOM ACCESS FILES

*You never know till you try to  
reach them how accessible men are;  
but you must approach each man by  
the right door.*

*Henry Ward Beecher, 1887*

## 5.0. OVERVIEW

This chapter introduces you to random-access text files. You will be shown how to create them, store information in them, and retrieve information from them. This chapter will assume that you have read the previous chapter on sequential-access text files. There are a number of similarities between these two types of files.

The first part of this chapter explains the structure of a random-access text file. You will be able to see the different ways the random-access and sequential-access text files organize data.

The last part of this chapter explains the syntax and options of each of the commands introduced. It should be pointed out that even though the commands introduced seem to be like those in the previous chapter, they really are different.

*You will notice that the commands presented here are very similar to those presented in the last chapter. However, the options for the commands are different.*

The commands introduced in this chapter are shown in the box below.

OPEN	e.g.,	OPEN EXAMPLES,L28 OPEN /MY/LIST,L200,S6,D1	Deferred mode only
READ	e.g.,	READ EXAMPLES	Deferred mode only
WRITE	e.g.,	WRITE EXAMPLES WRITE /MY/LIST,S6,D1	Deferred mode only
CLOSE	e.g.,	CLOSE CLOSE /MY/XMAS.LIST	Immediate and deferred
DELETE	e.g.,	DELETE /MY/XMAS.LIST	Immediate and deferred
APPEND	e.g.,	APPEND EXAMPLES,L28 APPEND /MY/LIST/L200,D1	Deferred mode only
FLUSH	e.g.,	FLUSH /MY/XMAS.LIST	Deferred mode only

## 5.1. RANDOM-ACCESS FILES

The random-access file is different from the sequential file in that you may access any individual record that has been written to the diskette. Further, the form of storage on the diskette is different. The file is made up of individual records. Each record contains fields and each field contains either characters, numbers or both. The hierarchy of this organization is shown in Figure 5.1.

This type of file has some other distinguishing characteristics. Each record is exactly the same size and length, and, in general, data stored in each record is in the same order and of the same composition. Each field of each record may be accessed randomly.

The random-access file structure is very useful, flexible, and extremely versatile. However, it is not necessarily efficient in conserving diskette storage space. So, for the gain in flexibility, usefulness, and power, you give up some storage space.

The random-access file structure requires that you specify additional file parameters. The first is the length specification. Since all records in the file are the same size, ProDOS allocates the specified record length space on the diskette each time you WRITE a new record to the file. Second, you must specify the record in the file you wish to READ or WRITE. When you OPEN communication with a random-access file, you specify the file length in addition to the file name. You also may specify the slot and drive; however, these are optional.

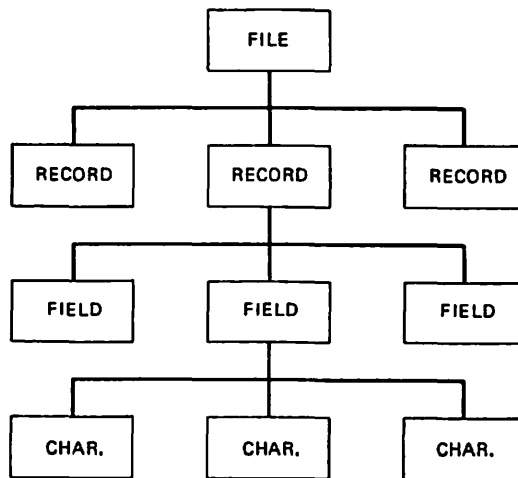


Figure 5.1. File hierarchy.

When you tell ProDOS what type of communication you wish to perform, either READ or WRITE, you then must specify the file name and the record number. Other options you may specify will be discussed later.

#### 5.1.1. The record length

The first time you open a random-access file, you *must* specify the record length. The length specification of a record represents the number of characters that may be stored in that record.

Let's look at an example of an OPEN statement.

```
255 PRINT D$;"OPEN /MY/BANK/NOTES/PAID ,L44"
```

Notice that the length of the record was specified, but the number of records in the file was not specified. You do not have to worry about that since ProDOS keeps track of that item plus the correct storage location of the records in the file.

#### 5.1.2. Writing a record

The writing of data to a record in a random-access file requires that you specify the record number to be written. If that record was previously written to, the data now written will overwrite all of the previous data. If the record specified has never been written to, ProDOS will reserve sufficient space on the diskette under the file name to store the new data. So, if you are only going to write one character to the PAID file, a total of 44 bytes of diskette storage will be set aside as being full for each record in the file.

Let's look at an example of the WRITE statement.

```
260 PRINT D$;"WRITE /MY/BANK/NOTES/PAID,R22"
```

The subsequent PRINT statements will actually store data into record number 22. You must specify a record number.

### 5.1.3. Record character storage

Let's now look at how data is stored into a random-access file. Assume that you have the program segment:

```
JNEW

JLIST

10 REM* RANDOM.NUMBERS PROGRAM
30 D$ = CHR$(4)
40 FOR I = 1 to 4: READ A$(I): NEXT I
42 DATA ONE, TWO, THREE, FOUR
   :

100 PRINT D$;"OPEN NUMBERS.RANDOM,L20"
110 FOR R = 1 TO 4
120 PRINT D$;"WRITE NUMBERS.RANDOM,R";R
130 PRINT A$(R)
140 NEXT R
150 PRINT D$;"CLOSE NUMBERS"
500 END
```

It is recommended that you power up your system with the /SCRATCH.DISK in the boot drive. When you have the Applesoft II BASIC prompt character and cursor you are ready to enter the program above. By the way, you will add to this program later.

<i>Lines</i>	<i>Description</i>
Line 10	Remark statement identifying program.
Line 30	The variable D\$ contains the CTRL-D ProDOS character that signifies the following instruction is a ProDOS command.
Line 40	FOR—NEXT I loop to READ data into the A\$ vector.
Line 42	Data statement corresponding to the READ in previous program line.
Line 100	OPEN communications with ProDOS. OPEN is discussed in Section 5.2.
Line 110	Top of the R FOR—NEXT loop.

Line 120    The communications is to WRITE data to the diskette at record number R.  
               WRITE is discussed in Section 5.4.  
 Line 130    Store each data element in correct record to the diskette.  
 Line 140    Range of the R loop. NEXT R.

*Line 130 does not have the CTRL-D preceding the operation. This is because all output has already been redirected to the file through the instruction in line 120.*

Line 150    Terminate communications with ProDOS. CLOSE is discussed in Section 5.5.  
 Line 500    END the program.

When you have this program entered, SAVE it as /SCRATCH.DISK/RANDOM.NUMBER and then RUN it. After running the program, the information you have stored on a diskette is stored in the following way under the name NUMBERS.RANDOM. This shown in Figure 5.2.

This particular file has four records that encompass 20 characters each for a total of 80 characters, including all of the carriage returns. Please notice that the first record in the file is designated 1. Further, notice that records 0 and 5 are missing. That is because those records were not written.

The next thing to realize is that the specified record length of 20 includes the final carriage return delimiter that separates individual records. You should take that into account when specifying the length size of a record.

The last item concerns a little bit of poetic license that has been taken. The carriage return character delimiters are shown at the end of each record when in reality it will be placed immediately after the last character written to that record. This was done for the purpose of easily visualizing how a record is stored on a diskette.

R#	Characters stored
1	ONE
2	TWO
3	THREE
4	FOUR

Notes: (1) The | character represents a carriage return.  
 (2) Numbers represent the record numbers.  
 (3) R# represents the record number column.

Figure 5.2. Random-access file storage.

The next obvious question that arises is: "How can I see what has been stored in the file?" Fair question. In the next few paragraphs more will be added to the program /SCRATCH.DISK/RANDOM.NUMBER.

Load the program /SCRATCH.DISK/RANDOM.NUMBER from your disk drive and LIST the program. Then enter the following additional code:

```
20 DIM A$(10)
   :
200 PRINT D$;"OPEN NUMBERS.RANDOM,L20"
210 FOR R = 1 TO 4
220 PRINT D$;"READ NUMBERS.RANDOM,R";R
230 INPUT A$(R)
240 NEXT R
260 PRINT D$;"CLOSE NUMBERS"
300 FOR I = 1 TO 4: PRINT A$(I): NEXT I
```

<i>Lines</i>	<i>Description</i>
--------------	--------------------

Line 20	Dimension the variable A\$ to handle 11 elements.
Line 200	OPEN communications with ProDOS. OPEN is discussed in Section 5.2.
Line 210	Top of the R FOR—NEXT loop.
Line 220	The communications is to READ data from the diskette. READ is discussed in Section 5.3.
Line 230	Read the records into the A\$ vector.
Line 240	Range of the R loop. NEXT R.

*Line 230 does not have the CTRL-D preceding the operation. This is because all input has already been redirected to the file through the instruction in line 220.*

Line 260	Terminate communications with ProDOS. CLOSE is discussed in Section 5.5.
Line 300	Print the contents of the A\$ vector to the video screen.

When you have the additional program code entered, SAVE it as /SCRATCH.DISK/RANDOM.NUMBER and then RUN the program again. If everything goes well, you should see displayed on the screen:

```
ONE
TWO
THREE
FOUR
```

```
]
```

You now know that what you stored may be retrieved and seen through the mechanism of supporting code that prints what has been retrieved from the diskette. Also, you have now written a very simple data storage and retrieval system. It really wasn't very difficult, was it?

If you now go back to Chapter 4 and look through Section 4.1, you will see that the two programs RANDOM.NUMBER and NUMBER.PROGRAM are very similar. This was done on purpose to give you a simple but graphic comparison. When running these two programs, the results are exactly the same, they both run in essentially the same amount of time, and occupy essentially the same space on a diskette.

So, what's the differences?

In these simple examples the differences are difficult to see immediately. The major differences are evident in the way disk I/O code must be written. In the next program example, the differences will become much more dramatic.

Up to this point, there has been only one field in each record of the random-access text file. There are usually multiple fields of data in one record. By this I mean that there are multiple pieces of data within a single record of a random-access file. This is done by placing commas between each field that makes up each record in the file. ProDOS does this for you. For example, let's put three fields of information within one record using the following code:

```
JNEW
```

```
JLIST
```

```
10 REM* MULTI-FIELDS PROGRAM
20 DIM A$(10)
30 D$ = CHR$(4)
   :
100 PRINT D$;"OPEN FIELDS.DATA,L20"
110 HOME: PRINT TAB( 11) "MULTI-FIELDS DEMO": PRINT
120 PRINT D$;"WRITE FIELDS.DATA,R1"
130 PRINT "FIRST": PRINT "SECOND": PRINT "THIRD"
135 PRINT D$;"WRITE FIELDS.DATA,R2"
140 PRINT "ELEMENTS": PRINT "PER": PRINT "RECORD"
150 PRINT D$;"CLOSE"
200 PRINT D$;"OPEN FIELDS.DATA"
210 PRINT D$;"READ FIELDS.DATA,R1"
230 INPUT A$(1),A$(2),A$(3)
240 PRINT D$;"READ FIELDS.DATA,R2"
250 INPUT A$(4),A$(5),A$(6)
260 PRINT D$;"CLOSE"
300 FOR I = 1 TO 6 : VTAB 2+I
310 PRINT A$(I)
320 NEXT I
500 END
```



Power up your system with the /SCRATCH.DISK in the boot drive. Then enter this program from the keyboard. When it has been entered, save the program using the name MULTI.FIELDS. Then RUN the program to see what happens.

You should get the following output:

### MULTI-FIELDS DEMO

```
FIRST
SECOND
THIRD
ELEMENTS
PER
FIELD
```

```
]
```

#### *Lines      Description*

Line 10	Remark statement identifying program.
Line 20	Dimension the variable A\$ to handle 11 elements.
Line 30	The variable D\$ contains the CTRL-D ProDOS character that signifies the following instruction is a ProDOS command.
Line 100	OPEN communications with ProDOS. OPEN is discussed in Section 5.2. Record length is 20.
Line 110	Clear the screen. HOME. Print screen title centered at the top. Print a blank line on the screen. PRINT.

*It is permissible to print to the screen after a file is open provided you do the printing before the WRITE command to the file is executed.*

Line 120	The communications is to WRITE data to the diskette. WRITE is discussed in Section 5.4. WRITE record 1.
Line 130	Store the first record to the diskette. This record contains three fields.
Line 135	The communications is to WRITE data to the diskette. WRITE record 2. WRITE is discussed in Section 5.4.
Line 140	Store the second record to the diskette. This field also contains three fields.

*Lines 130 and 140 do not have the CTRL-D preceding the operation. This is because all output has already been redirected to the file through the instructions in lines 120 and 135.*

Line 150	Terminate communications with ProDOS. CLOSE is discussed in Section 5.5.
Line 200	OPEN communications with ProDOS. OPEN is discussed in Section 5.2.

Line 210    The communications is to READ data from the diskette.  
             READ is discussed in Section 5.3. Read record 1.

Line 230    Read the first record from the diskette. This record contains three fields.

Line 240    The communications is to READ data from the diskette.  
             READ is discussed in Section 5.3. Read record 2.

Line 250    Read the second record from the diskette. This record also contains three fields.

*Lines 230 and 250 do not have the CTRL-D preceding the operation. This is because all input has already been redirected to the file through the instruction in lines 210 and 240.*

Line 260    Terminate communications with ProDOS. CLOSE is discussed in Section 5.5.

Line 300    Top of the FOR—NEXT I loop.  
             Position cursor on the screen. VTAB 2 + I.

Line 310    Print the contents of the A\$ vector to the video screen.

Line 320    Range of the I loop. NEXT I.

Line 500    END the program.

Notice that lines 130 and 140 of the program are actually written to the file without any commas embedded between fields in the record. Later in the program, you will need to retrieve the three fields of data in each record. Since INPUT statements consider commas as delimiters signifying the end of an element of data, it is necessary to use multiple data inputs. This has been shown in lines 230 and 250.

At this point, it might be a good idea to look at how these two multiple-field records are stored on a diskette. This is shown in Figure 5.3.

Notice that the extra space available in each record has been shown only to help you to visualize the structure. Notice that you should account for both the commas and the carriage return delimiter at the end of a record. The length parameter specified was set at 20. This is actually a little small, and was done for illustration purposes.

Now, by using the information presented, how would you read only the first field of each record? Another question could be: how could you use the GET instruction to retrieve information from the FIELDS.DATA file? I am sure you can think of many other possible questions.

Again, it would be a good idea to compare this last program with its counterpart in Section 4.1.

Characters stored:	FIRST,SECOND,THIRD	ELEMENTS,PER,FIELD
Record numbers:	{        1        }	{        2        }

Notes: (1) the | character represents a carriage return.  
 (2) Numbers represent the record numbers.

Figure 5.3. Random-access file character storage.

### 5.1.4. Reading from a record

As you saw in the last two small programs, the READ command for a random-access file requires that you specify the record number that is to be read. For example, look at line 210 from the last program example. The line is repeated here.

```
210 PRINT D$;"READ FIELDS.DATA,R1"
```

The following sections of this chapter explain each of the commands that are used with random-access files.

## 5.2. THE OPEN COMMAND

Before you are able to read or write data from or to any random-access file, you must first open communication with that text file. The open command will also set aside file buffer space in memory to handle temporary data storage. This command operates the same as in DOS 3.3 except for the length option default. The syntax for this command is:

```
]OPEN pn [,L#] [,S#] [,D#]
```

The first time you open communication with a random-access file, you must supply the length parameter option. The # in this option is the number of bytes that each record in the file may hold. After the file has been opened once, the length option is no longer needed because ProDOS assumes that the length is the same.

Opening a random-access file with a record length other than the length used to create the file, that length will be used until the file is closed. However, the original length still remains as the default record length.

When you open a text file, ProDOS sets aside a file buffer memory space that holds file identification information. Further, the system is prepared to either read or write data starting at the beginning of the opened file. ProDOS allows you to have 8 files open simultaneously.

When using files, it is always a good idea to FLUSH and CLOSE all open files before leaving the current program. It is possible you could loose data that you thought was stored to the diskette.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn indicates the name or partial pathname of the file to be opened. If the file already exists, it must not already be open. If the file does not already exist, it is created by ProDOS.
----	---

[,L#]	You are required to use the record length option the first time you open the file. This can be as early as when the file is created. If, however, you create the file
-------	---

without the length option, ProDOS assigns a record length of 1 for the file. A record length may be any positive integer in the range of 1 to 65535.

[,S#]

The slot and drive options have their normal meanings.

[,D#]

### *Examples for the OPEN command*

```
PRINT D$;"OPEN DUMMY.DATA,L128"
PRINT D$;"OPEN /MY/BANK/NOTES/PAID,L30,S6,D2"
PRINT D$;"OPEN PAID,L";L;"S";SL;"D";DR
PRINT D$;"OPEN ";F$;"L";LN
```

## 5.3. THE READ COMMAND

Before you can read any data from a file, you must identify to ProDOS that you want to read data. The READ command tells ProDOS the file name and the record number you want to read. This command works essentially the same as in DOS 3.3. The syntax for this command is:

```
]READ pn [,R#] [,F#] [,B#]
```

The READ command specifies to ProDOS:

- The file pathname.
- The record number.
- The position within the record.

The READ command will remain in effect until another ProDOS command is executed.

### *Option      Description*

pn	pn indicates the name or partial pathname of the file to be accessed.
[,R#]	# is the number of the record to which characters are to be stored. If this number option is omitted, record 0 is assumed by ProDOS. The maximum record number is 16 megabytes divided by the file's record length, or 65535, whichever is smaller.
	If # is larger than any previous record number, the ENDFILE column in the catalog changes.

- [,F#] # is the number of fields that ProDOS should read and discard. Reading fields is done by starting at the current position and reading characters counting the specified number of carriage returns. This option changes the file's current position.
- [,B#] # is the number of bytes, or characters, which ProDOS should read and discard. This option changes the position in the file relative to the file's current position.

#### *Examples for the READ command*

```
PRINT D$;"READ DUMMY.DATA,R22"
PRINT D$;"READ ";F$;"R";R
PRINT D$;"READ DUMMY.DATA"
PRINT D$;"READ /MY/BANK/NOTES/PAID,R22"
```

## 5.4. THE WRITE COMMAND

It is necessary to tell ProDOS you want to WRITE data to some file record before you use the PRINT statement to put characters into a random-access file.

The WRITE command specifies to ProDOS:

- The file pathname
- The recorded number
- The position within the record

The WRITE command will remain in effect until another ProDOS command is executed. The syntax of this command is:

```
]WRITE pn [,R#] [,F#] [,B#]
```

Each time you wish to write to a different record, you need to use the WRITE command before writing the new data to that new record. If you use the WRITE command without specifying a record number, record zero will be written. This command operates essentially the same as in DOS 3.3.

#### *Option      Description*

- pn      pn indicates the name or partial pathname of the file to be written.
- [,R#]    # is the number of the record to which characters are to be stored. If this number option is omitted, record 0 is assumed by ProDOS. The maximum record

number is 16 megabytes divided by the file's record length, or 65535, whichever is smaller.

If # is larger than any previous record number, the ENDFILE column in the catalog changes.

[,F#] # is the number of fields that ProDOS should read and discard. Reading fields is done by starting at the current position and reading characters counting the specified number of carriage returns. This option changes the file's current position.

[,B#] # is the number of bytes, or characters, which ProDOS should read and discard. This option changes the position in the file relative to the file's current position.

#### *Examples for the WRITE command*

```
PRINT D$;"WRITE DUMMY.DATA,R22"
PRINT D$;"WRITE ";F$;"R";R
PRINT D$;"WRITE DUMMY.DATA"
PRINT D$;"WRITE /MY/BANK/NOTES/PAID,R22"
```

## 5.5. THE CLOSE COMMAND

When finished with all of your disk processing, you should CLOSE all of the files that were open. It is necessary to properly close open files in order to ensure that all data has been written to the file and that all file buffer space is released.

]CLOSE [pn]

The CLOSE command without the pn option will close all files and release all file buffer space. This command operates exactly as in DOS 3.3.

#### *Option      Description*

pn	pn indicates the name or partial pathname of the file to be closed. It must be identical to the pn that you previously opened.
----	--

#### *Examples for the CLOSE command*

```
PRINT D$;"CLOSE"
PRINT D$;"CLOSE DUMMY.DATA"
PRINT D$;"CLOSE /MY/BANK/NOTES/PAID"
PRINT D$;"CLOSE ";F$
```

## 5.6. THE DELETE COMMAND

The purpose of the DELETE command is to enable you to get rid of any file on a diskette except the volume directory.

```
]DELETE pn [,S#] [,D#]
```

This command operates exactly as in DOS 3.3.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn indicates the name or partial pathname of the file to be opened. If the file already exists, it must not already be open. If the file does not already exist, it is created by ProDOS.
[,S#]	The slot and drive options have their normal meanings.
[,D#]	

### *Examples for the DELETE command*

```
PRINT D$;"DELETE"
PRINT D$;"DELETE DUMMY.DATA"
PRINT D$;"DELETE /MY/BANK/NOTES/PAID"
PRINT D$;"DELETE ";F$
```

## 5.7. THE APPEND COMMAND

The APPEND command is used for the purpose of writing data to the end of the present file. This command is actually three commands in one. APPEND opens the file, positions to the end of the file, and then writes the data to the file. The syntax of this command is:

```
]APPEND pn [,L#] [,S#] [,D#]
```

After APPEND has been executed, you just have to issue the PRINT statements needed to store data into the file at the current end of the file. This command is not supported for random-access files in DOS 3.3.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn indicates the name or partial pathname of the file to be accessed.
----	---

- [,L#] You are required to use the record length option the first time you open the file. This can be as early as when the file is created. If, however, you create the file without the length option, ProDOS assigns a record length of 1 for the file. A record length may be any positive integer in the range of 1 to 65535.
- [,S#] The slot and drive numbers have their normal meanings.
- [,D#]

*Examples for the APPEND command*

```
PRINT D$;"APPEND DUMMY.DATA,R22"
PRINT D$;"APPEND ";F$;" ,R";R
PRINT D$;"APPEND DUMMY.DATA"
PRINT D$;"APPEND /MY/BANK/NOTES/PAID,R22"
```

## 5.8. THE FLUSH COMMAND

The FLUSH command is used for the purpose of ensuring that all of the data in the file buffer area is written to the diskette. ProDOS will store up to 512 bytes of data in its file buffer before actually writing anything to the diskette media. This command is for the purpose of emptying the file buffer. The syntax of this command is:

]FLUSH [pn]

When you use the FLUSH command without the pathname option, all open files will be flushed. One caution: this command takes time and will slow down processing. This command is not supported by DOS 3.3.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn indicates the name or partial pathname of the file to be accessed.
----	---

*Examples for the FLUSH command*

```
PRINT D$;"FLUSH"
PRINT D$;"FLUSH DUMMY.DATA"
PRINT D$;"FLUSH /MY/BANK/NOTES/PAID"
PRINT D$;"FLUSH ";F$
```



## SUMMARY

This chapter discussed the random-access text file. The structure of the file and the way data is stored on a diskette were explained. Example programs were given, similar to those shown in Chapter 4. These examples were designed to show the differences in the program code written.

The characteristics of random-access file organization were shown. The method of data storage on a diskette within a random-access text file was also shown.

The tradeoffs between using sequential-access and random-access text files were discussed.

For each of the instructions discussed, a comparison was made to DOS 3.3 so that you can make the transition easily.

The commands introduced in this chapter were:

OPEN	e.g.,	OPEN EXAMPLES,L28 OPEN /MY/LIST,L200,S6,D1	Deferred mode only
READ	e.g.,	READ EXAMPLES	Deferred mode only
WRITE	e.g.,	WRITE EXAMPLES WRITE /MY/LIST,S6,D1	Deferred mode only
CLOSE	e.g.,	CLOSE CLOSE /MY/XMAS.LIST	Immediate and deferred
DELETE	e.g.,	DELETE /MY/XMAS.LIST	Immediate and deferred
APPEND	e.g.,	APPEND EXAMPLES,L28 APPEND /MY/LIST/L200,D1	Deferred mode only
FLUSH	e.g.,	FLUSH /MY/XMAS.LIST	Deferred mode only

## QUESTIONS

1. Describe how data is stored in a random-access file.
2. Compare sequential-access and random-access files. Account for both commands and storage differences.
3. Discuss the F# and B# options for those commands that have them.
4. When using random-access files, are you allowed to use variables within the commands? How do you do this?
5. What is the purpose of the APPEND command?
6. What is the purpose of the FLUSH command?

# 6. BINARY FILES

*Drown not thyself to  
save a drowning man.*

*Thomas Fuller, 1732*

## 6.0. OVERVIEW

This chapter will discuss the ProDOS commands that let you use and manipulate binary programs and files. Remember that the running of a binary program may be accomplished using the intelligent RUN command, —(DASH). That command was discussed in both Chapters 1 and 2.

The commands discussed in this chapter allow you to:

- load, save, and run binary programs.
- use binary programs to read and write characters.

The commands introduced in this chapter are shown in the box below:

BLOAD	e.g.,	BLOAD PIC1 BLOAD PIC1,A\$4000	Immediate and deferred mode
BSAVE	e.g.,	BSAVE PIC1,A\$2000 BSAVE PIC1,A8192	Immediate and deferred mode
BRUN	e.g.,	BRUN TONE.BEEP BRUN BEEPER, A\$300 — TONE.BEEP	Immediate and deferred mode
PR#	e.g.,	PR#1 PR#0	Immediate and deferred mode
IN#	e.g.,	IN#2 IN#0	Immediate and deferred mode

## 6.1. BINARY FILES

ProDOS, like DOS 3.3, allows you to store and retrieve portions of memory either to or from a diskette. Earlier, in Chapter 2, you were introduced to the RUN, LOAD, and SAVE commands. These commands dealt with Applesoft II BASIC programs. When dealing with binary files, the BRUN, BLOAD, and BSAVE commands perform similar functions. The major difference is that these commands deal with uninterpreted information stored in some portion of your computer's memory.

The B letter preceding the command names signifies that the file to be manipulated is binary in nature. Each command manipulates the information between the computer memory and the file storage location on a byte-for-byte basis. These commands are referred to as binary commands. Normally, these binary commands will involve a machine-language program, a high-resolution picture for one or both of the Apple II computer graphics screens, or any information in binary form that is in memory or stored on a diskette.

### 6.1.1. Binary addresses

If you are only going to run machine-language programs that are already stored on a diskette, you will not have to understand the memory organization of your computer. However, in all likelihood, your involvement with binary files will be much more extensive.

If you are going to save binary memory information, graphics screen data, or work with machine-language files, you will need to know how memory information is organized.

Memory is a continuous sequence of byte locations, each having a unique address. Each address is different from all other addresses so that there is no confusion when referencing any particular location. This is analogous to your home address. It is different from your neighbors' addresses. The first memory location in your computer is 0 in decimal which equals \$0000 hexadecimal. The next memory location is 1 in decimal, \$0001 in hexadecimal. Therefore, a 64K Apple computer has as its highest memory location 65535 in decimal, \$FFFF in hexadecimal. Appendix J gives you a decimal to hexadecimal conversion table.

In general, when working with binary files, you must specify a number of different memory locations to save any portion of memory to a diskette file. First of all, you need to define the starting address of the file in memory. Next, you need to define the length of the file to be saved. This may be done in either of two ways. You may use the length option or the ending address option. Further, you have the option of specifying these options in either decimal notation or hexadecimal notation. In fact, you may mix up these option notations. These options will be discussed in the next section.

### 6.1.2. Command options

When saving binary information, it is necessary to define the address of the first memory location to be saved. This is determined by using the **A#** option. The number of memory locations to be saved must also be defined using the **L#** option.

The **A#** and **L#** options may be expressed in either decimal or hexadecimal notation. Both of these options are the same as in DOS 3.3.

Alternatively, you may use the new **E#** option, which is the ending address for the binary file to be transferred. This is a new option in ProDOS. The length of a binary file may be calculated by subtracting the starting address from the ending address. Caution: Do not mix notation values. Subtract decimal values from decimal values and hexadecimal values from hexadecimal values.

All of these options are discussed in detail in Section 6.3.

## 6.2. THE BLOAD COMMAND

This command causes binary data to be transferred from any disk file to a specified portion of memory. Binary data is normally a machine-language program, a picture, or other graphics data. This command works the same as in DOS 3.3 except for the additional options.

The syntax for this command is:

**[BLOAD pn [,A#] [,B#] [,L#|E#] [Ttype] [,S#] [,D#]**

The vertical line signifies that either option may be used but both are not required.

The BLOAD command may be used to:

- transfer a machine-language program from file storage to memory.
- move a graphics image from file storage to the graphics screen.
- move any binary image from file storage to memory.

If you use this command with nothing but the pathname option, then the entire file at the default address values will be loaded from file storage to memory.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn is the pathname or partial pathname to indicate the file desired. If this is the only option specified, the entire file specified will be placed into memory starting at the stored default address. If you have the 80-column capability activated and use the CATALOG command, the starting address is presented in hexadecimal notation with an A preceding the address. This is shown in the SUBTYPE column.
[,A#]	This is the memory address at which the first byte of the file is to be loaded into memory. The address specified must be a valid memory location.
[,E#]	E# is the last memory address location to be transferred.
[,L#]	L# is the number of bytes to be transferred.

- [,B#] This option designates the first byte in the file that is to be transferred from the diskette to memory. If this option is not specified, then byte zero (\$0000) starts the transfer.
- [Ttype] Type is the three-letter abbreviation that indicates the type of file to be transferred. If no type is specified, the file will be the BIN type. See Chapter 3 for all of the abbreviations.
- [,S#] The slot and drive options have their normal meanings.
- [,D#]

When you are storing files on a diskette using ProDOS, the command used to save the file, SAVE or BSAVE, determines the file's storage format. An Applesoft II BASIC program is saved by using tokens that represent the language's keywords. A token is an ASCII numeric value that stands for a language keyword or symbol. For example, the keyword PRINT, is represented by the token value 186 = \$BA.

It is now possible to BLOAD that Applesoft II BASIC program into memory by using BLOAD with Ttype option as BAS. You could now make changes, look at the uninterpreted code, or do anything else. When you are finished, then BSAVE the file back onto the diskette, if necessary.

#### *Examples of BLOAD command*

```
BLOAD PICTURE
BLOAD PICTURE,A8192,E16383
BLOAD PICTURE,A$2000,L$2000,S6,D2
BLOAD PICTURE,A$2000,L8192
BLOAD PICTURE,A8192,A$2000
BLOAD PICTURE,B0,BIN,S6,D2
```

### **6.2.1. Installing machine-code routines**

Because of the way ProDOS dynamically allocates and deallocates memory for file buffers, it is somewhat more difficult to manage memory and guarantee which parts of memory will be free to store and protect machine-language routines.

For example, when you OPEN a file under control of ProDOS, the HIMEM pointer is moved down in memory by 1K. This 1K area of memory is used as a file buffer in the area where the old HIMEM used to be located. Further, that 1024 byte memory area, 4 pages, is marked as used by the system bit map. More will be said about this in Chapter 9.

Therefore, in order to store a machine-language routine into memory and protect that routine's memory area, you must also do the same as ProDOS. When moving HIMEM down, you must move HIMEM in increments of 256 bytes, a page at a time. Then you can BLOAD your routine into memory and, finally, mark the appropriate system bit map as used.

*You should do all machine-language loading into memory BEFORE you open any files. In this way, all file buffers will then reside below all of your own routines.*

### 6.3. THE BSAVE COMMAND

This command causes binary data to be transferred from a specified portion of memory to any type of diskette file. Data anywhere in the computer's memory may be transferred to a file on a diskette. This command works the same as in DOS 3.3 except for the additional options.

The syntax for this command is:

**]BSAVE pn ,A# ,L#|,E# [,B#] [,Ttype] [,S#] [,D#]**

The vertical line signifies that either option may be used but both are not required.

You may use the BSAVE command to:

- transfer a machine-language program from memory to a storage file.
- move a picture from a graphics screen to a storage file.
- move any portion of memory into any type of storage file.

When using the BSAVE command, it is necessary to use the pn, A#, and either the L# or E# options.

<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn is the pathname or partial pathname to indicate the file desired. If this is the only option specified, the entire file specified will be placed into memory starting at the stored default address. If you have the 80-column capability activated and use the CATALOG command, the starting address is presented in hexadecimal notation with an A preceding the address. This is shown in the SUBTYPE column.
,A#	This is the memory address at which the first byte of the file is to be saved from memory. The address specified must be a valid memory location.
,E#	E# is the last memory address location to be transferred.
,L#	L# is the number of bytes to be transferred.
[,B#]	This option designates the first byte in the file that is to be transferred from the diskette to memory. If this option is not specified, then byte zero (\$0000) starts the transfer.
[Ttype]	Type is the three-letter abbreviation that indicates the type of file to be transferred.

If no type is specified, the file will be the BIN type. See Chapter 3 for all of the abbreviations.

[,S#]     The slot and drive options have their normal meanings.  
[,D#]

#### *Examples of BSAVE command*

```
BSAVE HR1.PIC,A8192,L8192
BSAVE HR1.PIC,A$2000,L$2000
BSAVE HR1.PIC,A8192,L$2000,B0,S6,D2
BSAVE HR1.PIC,A$2000,L$2000
BSAVE HR1.PIC,A8192,E16383
BSAVE HR1.PIC,A16384,E24576,B0,S6,D2
```

## 6.4. THE BRUN COMMAND

This command causes binary data to be transferred from a binary disk file (BIN) to a specified portion of memory, and then executed. You will use this command to execute a binary program stored in a binary file on a diskette. This command works the same as in DOS 3.3 except for the added options.

The syntax for this command is:

]BRUN pn [,A#] [,B#] [,L#|,E#] [,S#] [,D#]

When you use this ProDOS command, the binary file specified by the pn is loaded into the memory locations specified by the options or the defaults and then commences execution of the program. The vertical line signifies that either option may be specified if you use that option but both are not required.

There is one caution when using this command. ProDOS does not differentiate between binary programs and binary data. Therefore, it is recommended that you use file names that relate to the contents of the file. For example, MUSIC.PROG to designate a binary program that might play musical tunes or HR2.PIC to designate a high-resolution page 2 graphics picture. There is the possibility that running a binary data or graphics picture could cause changes to ProDOS and ruin your whole day.

*As was shown in Chapter 2, you may use the — (DASH) command to run a machine-language program.*



<i>Option</i>	<i>Description</i>
---------------	--------------------

pn	pn is the pathname or partial pathname to indicate the file desired. If this is the only option specified, the entire file specified will be placed into memory starting at the stored default address. If you have the 80-column capability activated and use the CATALOG command, the starting address is presented in hexadecimal notation with an A preceding the address. This is shown in the SUBTYPE column.
[,A#]	This is the memory address at which the first byte of the file is to be loaded into memory. The address specified must be a valid memory location.
[,E#]	E# is the last memory address location to be transferred.
[,L#]	L# is the number of bytes to be transferred.
[,B#]	This option designates the first byte in the file that is to be transferred from the diskette to memory. If this option is not specified, then byte zero (\$0000) starts the transfer.
[,S#]	The slot and drive options have their normal meanings.
[,D#]	

*Examples of BRUN command*

```
BRUN MUSIC.PROG
— MUSIC.PROG
BRUN MUSIC.PROG,A$300
BRUN MUSIC.PROG,A768,B0,S6,D1
BRUN MUSIC.PROG,A$300,B0,L30,S6,D1
```

**6.5. THE PR# AND IN# COMMAND**

These commands normally are used to redirect your computer's output or input. This was discussed previously in Chapter 3. These commands allow you to communicate through the expansion slots in addition to the normal forms of communication. These commands work the same as in DOS 3.3 except for the additional options.

The syntax for these commands is:

]PR# slot [,A#] A#
]IN# slot [,A#] A#

The vertical line signifies that either option may be used but both are not required.

<i>Option</i>	<i>Description</i>
---------------	--------------------

slot	This value may be any integer number from 0 through 7. If the number is zero, then output is to the video screen. If slot is any number from 1 through 7, then input and output are redirected to the slot specified.
[,A#]	# is an address option. This is for the purpose of defining the address of the character input or output routine you wish to use.
A#	Same as above.

The A option is an added capability to the PR# and IN# commands that was not present in DOS 3.3. This capability allows you to write your own character input routines or output routines. Then when you reference that particular slot, the routine stored at the A option address will actually be executed.

Another very handy capability allows you to change the physical address mapping of the expansion slots.

For example, you could reassign slot addresses.

```
]PR# 1,A$C200
```

In this case you have reassigned slot 2 to slot 1. Since the normal slot for a printer is slot 1 and you have a printer in slot 2, you have actually reassigned slot numbers.

*Examples of PR# and IN# commands*

```
PRINT D$;"PR#0"
PR#6
PR#1
PRINT D$;"PR#1"
IN#2,A$C200
IN#1,A$C200
PRINT D$;"IN#2"
IN#0
PRINT D$;"IN#3"
```

## 6.6. THE MONITOR AND PRODOS

Since this chapter deals with binary files, now would be a good time to discuss a very interesting new capability of ProDOS. There are probably times when you will be in the monitor and you wish to know what has been stored on a diskette. To get into the monitor, type:

**]CALL —151**

Then you will be presented with the monitor prompt;

\*

All of the ProDOS commands will still work from within the monitor. For example, you may wish to type:

**\*CAT**

This will give you the normal 40-column catalog display. From here you may run an Applesoft II BASIC program, perform a warm boot, return to the BASIC prompt, or other monitor evolutions.

Normally, you will want to return to the monitor by typing:

**\*CTRL-C**

This means to type the CTRL key and the C key simultaneously, followed by typing the RETURN key.

**SUMMARY**

This chapter explained and discussed in detail those ProDOS commands that allow you to manipulate binary files.

The general organization of memory was discussed, and the use of the address options used with the binary commands was shown.

The commands introduced in this chapter were:

BLOAD	e.g.,	BLOAD PIC1	Immediate and deferred mode
		BLOAD PIC1,A\$4000	
BSAVE	e.g.,	BSAVE PIC1,A\$2000	Immediate and deferred mode
		BSAVE PIC1,A8192	
BRUN	e.g.,	BRUN TONE.BEEP	Immediate and deferred mode
		BRUN BEEPER,A\$300	
		— TONE.BEEP	
PR#	e.g.,	PR#1	Immediate and deferred mode
		PR#0	
IN#	e.g.,	IN#2	Immediate and deferred mode
		IN#0	

## QUESTIONS

1. If you did not have the FILER program, how could you move the PRODOS, BASIC.SYSTEM, and STARTUP programs to a newly formatted diskette?
2. What does the E# option mean? Where might this information be helpful? Where is this information displayed?
3. How is the Ttype option helpful? Where and when is it used?
4. How can you redirect or remap a slot assignment?
5. How do the IN# and PR# commands work?

# 7. EXECUTIVE FILES

*Philanthropic and religious bodies do  
not commonly make their executive  
officers out of saints.*

*Emerson, 1860*

## 7.0. OVERVIEW

This chapter explains the EXEC command. This command allows you to have the Apple computers take control of operations through the use of sequential-access text files. The sequential-access text file may contain:

- ProDOS commands,
- Applesoft II BASIC program, or
- Input statements.

It is possible, through the use of the EXEC command, to:

- Convert Integer BASIC programs to Applesoft II BASIC programs.
- Repair programs.
- Insert routines into programs.

- Renumber programs or portions of programs.
- Move code from programs to text files to an APPLEWRITER file. (See the APPLEWRITER EXTENDED program.)
- Create an automatic, turnkey set of routines that run without intervention by you, the operator.

The command introduced in this chapter is shown in the box below:

EXEC	e.g.,	EXEC DUMMY	Immediate and deferred mode
------	-------	------------	-----------------------------

## 7.1. EXEC FILES DEMONSTRATION

There are actually two steps involved in the creation of an EXEC file. These are:

1. Create and RUN an Applesoft II BASIC program that creates an EXEC file.
2. Use the EXEC command to perform the functions of the EXEC file commands. Commands are taken from the EXEC file.

The Applesoft II BASIC program you create must do the following:

- OPEN a sequential-access text file,
- WRITE to or APPEND to a file,
- put commands into the text file using the PRINT or LIST command,
- CLOSE the sequential-access text file.

Let's create a small program to illustrate what is required to be done. Power up your system with the SCRATCH.DISK in the boot drive and then enter the following small program.

```
JNEW
```

```
JLIST
```

```
10 REM * BASIC PROGRAM
100 HOME : PRINT TAB( 16) "EXECS IT": PRINT
110 PRINT "ONCE UPON A TIME"
120 PRINT "THERE WAS AN EXEC"
130 PRINT "WHO RAN EVERYTHING"
140 PRINT "WITHOUT HELP FROM YOU"
150 END
```

Now that you have entered this program, save it as EXEC.PROG to the SCRATCH.DISK. You could use the command:

```
JSAVE /SCRATCH.DISK/EXEC.PROG
```

Since your system is still up and running, you next write the second program that will, when RUN, create the text file that will later be EXECuted. The program to be entered is:

```
JNEW
```

```
JLIST
```

```
10 D$ = CHR$ (4)
100 PRINT D$;"PREFIX /SCRATCH.DISK/"
110 PRINT D$;"OPEN DO.EXEC"
120 PRINT D$;"WRITE DO.EXEC"
130 PRINT "PREFIX /SCRATCH.DISK"
140 PRINT "CAT"
150 PRINT "RUN EXEC.PROG"
160 PRINT "LIST"
170 PRINT D$;"CLOSE DO.EXEC"
180 END
```

Now that you have this program entered, save it as EXEC.IT. This can be done by:

```
JSAVE /SCRATCH.DISK/EXEC.IT
```

You now have the two required programs saved on the SCRATCH.DISK. The next thing to do is to:

```
JRUN /SCRATCH.DISK/EXEC.IT
```

which will create the sequential-access text file named DO.EXEC. Finally you are ready to use the EXEC command. Type the command:

```
JEXEC DO.EXEC
```

This will cause the commands in the sequential-access file named DO.EXEC to be executed. You now have a program named EXEC.IT which may be used to create EXEC files of all kinds.

The next section will discuss the details and options of the EXEC command.

## 7.2. THE EXEC COMMAND

The EXEC command allows you to take commands and data from a sequential-access text file instead of from the keyboard or through other text file forms. This command works essentially the same as it does in DOS 3.3. The syntax for this command is:

```
]EXEC pn [,F#] [,S#] [,D#]
```

There are some interesting things that should be known when an EXEC file is actively running. These are:

The program is not affected by either a  
NEW command or  
CLOSE command.

The program cannot be stopped by a CTRL-C.

Monitor commands cannot be executed from within an EXEC file.

If an EXEC file is executing:

It may execute an Applesoft II BASIC program.

Interrupting the program with CTRL-C, will usually interrupt the EXEC file.

Subsequent INPUTted data from program will be taken from the EXEC file.

It may execute another EXEC file.

The second EXEC file will replace the first EXEC file.

### *Option    Description*

pn	pn is the pathname or partial pathname that identifies the EXEC file.
[,F#]	# is the number of fields to skip at the beginning of the EXEC file. This is accomplished by counting the number of instruction delimiters passed over.
[,S#]	The slot and drive options have their normal meanings.
[,D#]	

### *Examples for the EXEC command*

```
EXEC DO.EXEC
EXEC DO.EXEC,F2
EXEC /SCRATCH.DISK/DO.EXEC
EXEC /SCRATCH.DISK/DO.EXEC,S6,D1
```



### 7.3. EXEC USES

One of the more interesting and useful applications that the EXEC command may accomplish is the transforming of an Applesoft II BASIC program to a text file. Why would you want to do that? Once an Applesoft II BASIC program has been captured into a sequential-access text file you can:

- edit the program using a word processor, such as APPLEWRITER II or APPLEWRITER IIe.
- merge parts of one program into another program, such as your favorite subroutines.
- insert subroutines from a subroutine file into a large program.
- connect two or more programs together.
- convert an Integer BASIC program to Applesoft II BASIC program.

Probably the most powerful use of the EXEC command is the combination of capturing an Applesoft II BASIC program using EXEC and performing global edits, changes, and additions to that program using APPLEWRITER. When you have finished with the new word-processor file and saved the latest version to diskette, then EXEC your new program.

### SUMMARY

This chapter discussed only the one command, EXEC.

An entire chapter was devoted to this command because it is very powerful and allows you to accomplish many very interesting, useful things.

A number of possible uses for the EXEC command were discussed. However, not all of the possibilities were explained or discussed. Your imagination should provide you with many other possibilities.

### QUESTIONS

1. Discuss the requirements for creating an EXEC file.
2. Discuss various ways that you could use the EXEC command.
3. Discuss in detail how you would use EXEC operating in ProDOS with APPLEWRITER II operating in DOS 3.3. Include how to go from DOS 3.3 to ProDOS.
4. Discuss how to convert an Integer BASIC program to Applesoft II BASIC.
5. How would you insert a series of subroutines into an Applesoft II BASIC program using the EXEC command?

# 8. THE PRODOS FILER AND CONVERT PROGRAMS

*Do we move ourselves, or are moved*

*by an unseen hand at a game?*

*Alfred Lord Tennyson, 1865*

## 8.0. OVERVIEW

This chapter discusses the two major utility programs included in ProDOS. Their functions are to perform many of the housekeeping, organization, conversion, and general system operation tasks. Without these programs, the day-to-day operation of your system would be far more ineffective and less productive. For those of you who are familiar with DOS 3.3, there are two programs provided on the SYSTEM MASTER diskette that are direct corollaries to these two expanded utilities in ProDOS. Figure 8.1 shows these program sets:

ProDOS	DOS 3.3
<hr/>	<hr/>
FILER	FID and COPYA
CONVERT	MUFFIN

Figure 8.1. ProDOS-DOS programs.

The /PRODOS/FILER program performs all of the functions that the FID program performed plus a number of others. The /PRODOS/FILER program allows you to organize the information stored on a diskette. The /PRODOS/FILER will probably be used more than any of the other programs on the EXAMPLES disk. This program is covered in the first part of this chapter.

Throughout the discussion of the /PRODOS/FILER program, there are a number of ProFile notes. ProFile is the name of Apple's hard disk mass storage device. These notes are for the purpose of helping you use the /PRODOS/FILER and the ProFile together.

The /CONVERT program allows you to convert all of your DOS 3.3 programs to PRODOS. This is similar to the function of the MUFFIN program that converts DOS 3.2.1 programs and files to DOS 3.3 programs and files. However, /CONVERT is much more capable than MUFFIN.

Below are presented the keystroke menu selections for the /PRODOS/FILER and /CONVERT programs to give you a quick summary of the keystrokes required for both the /PRODOS/FILER and /CONVERT programs.

Those keystroke options that are defined in lowercase characters are for the /CONVERT program and those in uppercase are for the /PRODOS/FILER program.

#### ? - TUTOR

This keystroke gives you information about that part of the PRODOS/FILER you are currently using. This also explains how to type keyboard entries.

#### A - ALTER WRITE-PROTECTION

This keystroke allows you to alter, or change, the protection on a file.

#### B - BLOCK ALLOCATION

This keystroke lets you see the total number of blocks on a volume, how many are used, and how many are still available for storage.

#### C - COPY A VOLUME

This keystroke allows you to make an exact copy of an entire volume onto another volume.

#### C - COPY FILES

This keystroke allows you to create an exact duplicate of individual files from one diskette to another.

#### C - CHANGE SLOT AND DRIVE

This option allows you to change the slot-drive combination for the transferring of files.

#### D - CONFIGURATION DEFAULTS

This keystroke shows you the current defaults in effect.

**D - DETECT BAD BLOCKS**

This keystroke allows you to scan a volume for possibly damaged blocks. Damaged blocks could cause a loss of stored data.

**D - DELETE FILES**

This keystroke deletes files from a diskette without affecting the rest of the files on the volume.

**D - SET PRODOS DATE**

This option allows you to enter a date for date-stamping files.

**F - FILE COMMANDS**

This keystroke allows you to pick the file commands from the main filer menu.

**F - FORMAT A VOLUME**

This keystroke lets you format a new blank diskette for the storage of programs and files.

**K - COMPARE VOLUMES**

This keystroke lets you compare two volumes to determine if they are exact copies of each other.

**K - COMPARE FILES**

This keystroke gives you a byte-for-byte comparison of any two files that are named.

**L - LIST VOLUMES**

This keystroke lists the volumes that are currently active on your system.

**L - LIST PRODOS DIRECTORY**

This keystroke lists all of the files in the directory you name. You will be given the file's:

- type                      - write-protect status
- file size                - change date
- free blocks

**M - MAKE DIRECTORY**

This keystroke lets you create subdirectories on a diskette.

**P - SET PREFIX**

This keystroke lets you change or designate a pathname or partial pathname as the currently active PREFIX.

**P - SET PRODOS PREFIX**

This option allows you to set a new prefix pathname.

**Q - QUIT**

This keystroke lets you terminate the operation of the PRODOS/FILER program.

**R - RENAME A VOLUME**

This keystroke allows you to rename a volume without changing the contents.

R - RENAME FILES	This keystroke allows you to rename a stored file.
R - RESTORE DEFAULTS	This keystroke restores your system to the predefined default values.
R - REVERSE DIRECTION	This option allows you to reverse the direction of the transfer when you are converting files.
S - SELECT DEFAULTS	This keystroke lets you define those defaults peculiar to your own system.
T - TRANSFER FILES	This option allows you to actually transfer file to ProDOS.
V - VOLUME COMMANDS	This keystroke allows you to pick the volume subprograms menu.

## 8.1. USING THE FILER PROGRAM

In order to use the /PRODOS/FILER program, it is only necessary to boot the ProDOS disk and select the F (/PRODOS/FILER) option from the main menu screen. Remember that you do not need to type the RETURN key when making a selection. This will clear the screen, bring the /PRODOS/FILER program into memory, and automatically execute the program.

Once the program is executing, it is only necessary to follow the screen options to accomplish what is desired.

### 8.1.1. ProDOS FILER menu

The /PRODOS/FILER program has a number of subprograms that operate either upon a diskette as a whole or upon the individual files stored on a diskette. Collectively, these programs are normally called utility programs. Figure 8.2 shows these main program categories.

In a number of the subprogram screens, default values are shown. In those cases, you may select the defaults by simply typing the RETURN key. If you want to change the default value, simply type your required value.

One of the handy features of the /PRODOS/FILER program is the use of the ESCAPE key. You can use this key to restart a screen from the beginning, return to a previous screen, or back out of a selection. This allows you to be able to change subprograms and screens easily.

If you make a typing error and enter a keystroke that does not correspond to one of the legitimate commands, the /PRODOS/FILER program is very forgiving and allows you to reenter the correct keystroke.

By selecting the ? (TUTOR) option, you will be given information about the individual filer commands and terminology definitions that may be unfamiliar to you, at least for now. Section 8.1.2 discusses the TUTOR.

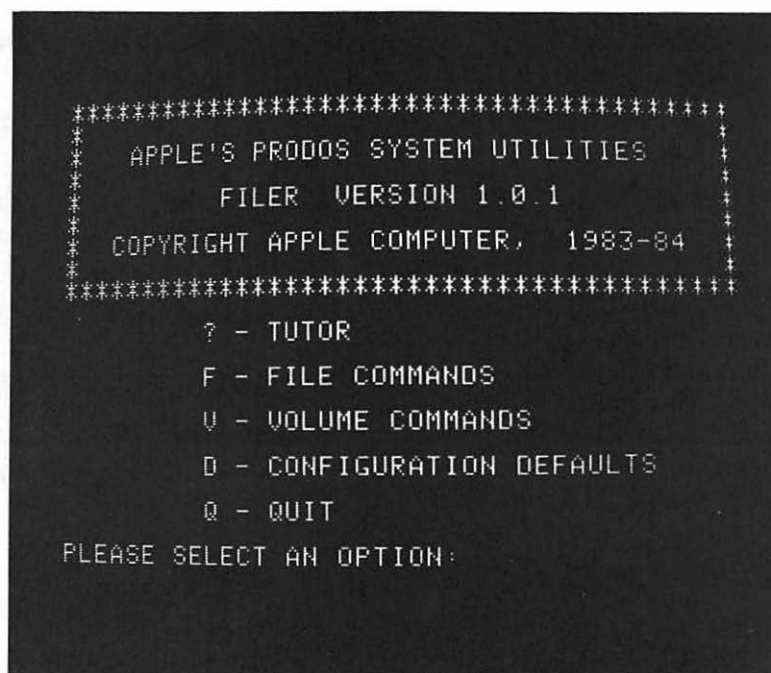


Figure 8.2. FILER main screen.

File commands are those that will affect only an individual file. Section 8.1.3 contains explanations of these subprograms.

Volume commands are those that affect a diskette as a whole unit. Section 8.1.4 discusses these subprograms.

Section 8.1.5 discusses the CONFIGURATION DEFAULTS option that lets you customize the defaults to match your particular system.

The QUIT option allows you to return to the USER'S DISK main menu or any other diskette of your choosing. This is discussed in Section 8.1.6.

### 8.1.2. TUTOR

The ProDOS TUTOR screens give you information quickly so that you do not have to read a book like this one to remember what is required. The ProDOS TUTOR is entered by typing the ? (QUESTION MARK) key on the keyboard.

As you proceed through this chapter, you will notice that every menu screen has a tutor option. The use of the tutor is very straightforward. Make the selection using the ? (QUESTION MARK) key and follow the continuation prompting at the bottom line of each screen.

### 8.1.3. File commands

This section will explain all of the file commands. This set of subprograms works with individual files stored on a diskette.

Your computer does not name files, you do. There are a number of rather simple rules for creating names for files. These rules have been covered earlier.

Figure 8.3 gives the FILE COMMANDS video display screen that is first presented when the F option is selected from the main /PRODOS/FILER menu.

In the following subsections, each menu option display screen will be shown and discussed.

**LIST PRODOS DIRECTORY.** This option allows you to list the files stored in any directory or on a diskette. When you select this option and enter the directory pathname, you will be presented with the files stored in that directory plus the following information:

- the directory name
- the type of file
- the size of each file
- the write-protect status

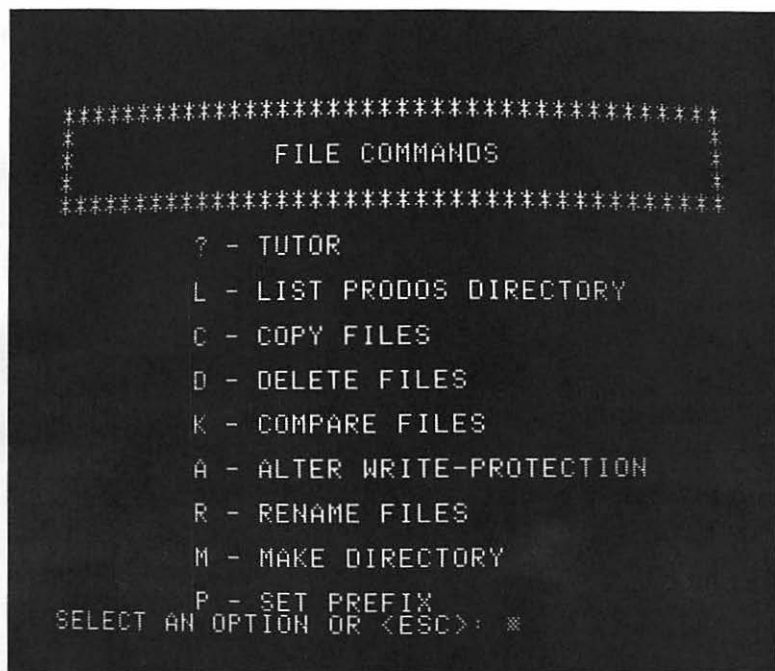


Figure 8.3. FILE COMMANDS screen.

- modification date
- the free blocks on the diskette
- the use blocks on the diskette

This screen is shown in Figure 8.4.

The data presented to the screen contains all of the files on the level specified and below.

*Notes:*

1. If the prefix is set to the directory you want to list, type the = (EQUAL) and press RETURN key.
2. If you type the ? (QUESTION MARK) you will be given only the used and free blocks on the diskette.
3. You may also get a listing of the directory on a printer by changing the display device. See the Configuration Defaults in Section 8.1.5.

*ProFile Note:*

Make sure that the prefix is set to the correct device name = /PROFILE.

```

*****
*          LIST PRODOS DIRECTORY          *
*                                          *
**PREFIX: /PRODOS/*****
--DIRECTORY--
  PATHNAME: (/PRODOS
)

--ENTER PATHNAME AND PRESS <RET>--

```

Figure 8.4. LIST PRODOS DIRECTORY screen.



**COPY FILES.** This option allows you to copy files from one diskette to another or from one directory to another. This screen is shown in Figure 8.5.

When the destination pathname already has an existing file stored, you will be asked:

DELETE EXISTING FILE? (Y/N)

You now have three options, These are:

- answer Y to replace existing file.
- answer N to leave existing file alone.
- cancel operation by pressing ESCAPE key.

*Notes:*

1. If you have a one-drive system, you will be prompted each time you need to change diskettes.
2. You can not copy files to a new subdirectory without first creating that directory. Use the make directory option first.

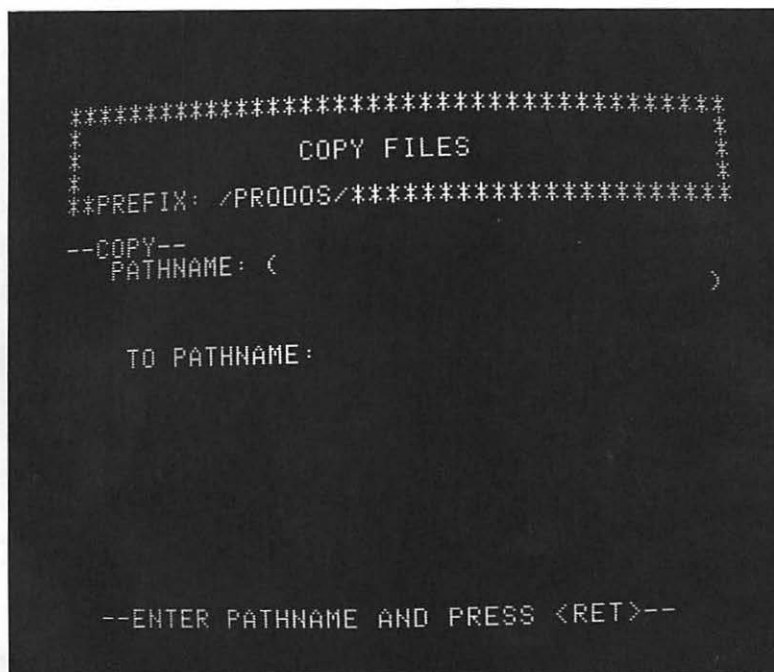


Figure 8.5. COPY FILES screen.





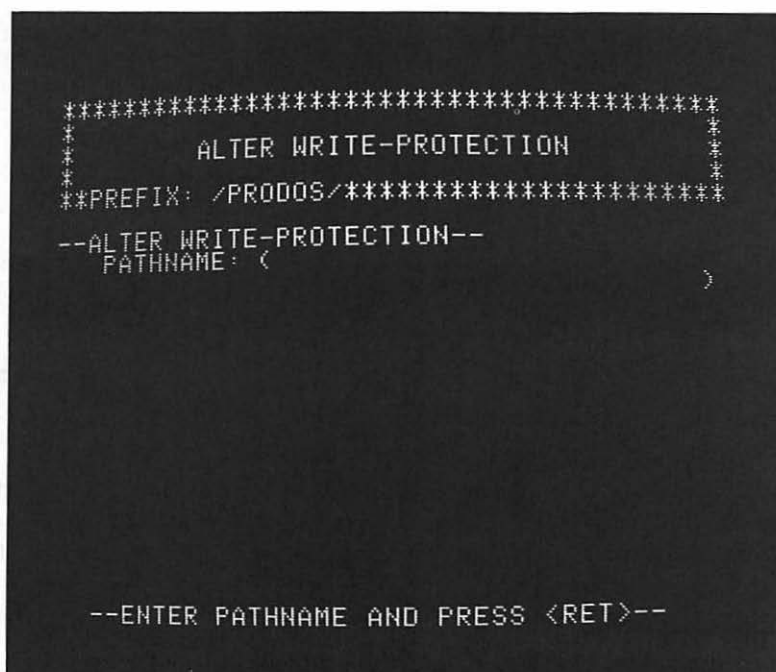


Figure 8.8. ALTER WRITE-PROTECTION screen.

either this option or the **CREATE** command within a program. A directory or subdirectory must be created in some fashion before you may store files in them.

When creating a directory on a diskette, the **/PRODOS/FILER** program will first check to determine if there is enough room on the diskette to hold the directory. This does not mean that there will be any room for the file; it just means that there is enough room for the directory file.

**SET PREFIX.** This option allows you to set a new prefix on a diskette. Setting a prefix means storing a pathname or partial pathname. Once you have set the prefix, it is not necessary to type that pathname again when accessing files. Once the prefix is set, it remains in effect until you change it or turn off the computer.

Notice that the current **PREFIX** is at the top of the display.

Anytime you type a pathname that does not begin with a slash (/) the current pathname the prefix will be used as the first part of the pathname. If you want the entire pathname ignored, then type in the full pathname. After you have finished typing the pathname, type the **RETURN** key to signify the completion.



```

*****
*                                     *
*               MAKE DIRECTORY       *
*                                     *
**PREFIX: /PRODOS/*****
--MAKE DIRECTORY--
  PATHNAME: (
)

--ENTER PATHNAME AND PRESS <RET>--

```

Figure 8.10. MAKE DIRECTORY screen.

```

*****
*                                     *
*               SET PREFIX           *
*                                     *
**PREFIX: /PRODOS/*****
--SET PREFIX--
  NEW PREFIX: (%PRODOS/
)

--ENTER PATHNAME AND PRESS <RET>--

```

Figure 8.11. SET PREFIX screen.

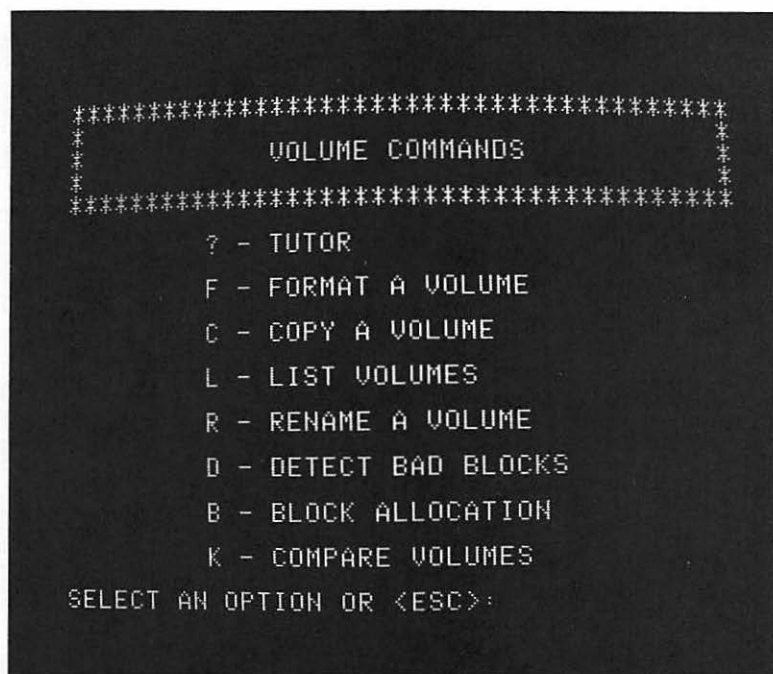


Figure 8.12. VOLUME COMMANDS screen.

Listing of the files on a volume may be accomplished by using the L (LIST VOLUMES) option. The video screen for this is shown in Section 8.1.4.3. This subprogram has a corollary in the FID program of DOS 3.3. Section 8.1.4.4 shows you the display that lets you RENAME a volume.

The next section describes how to detect bad blocks that might be present on a diskette. Section 8.1.4.6 gives you the block allocation present in a diskette.

The last section (8.1.4.7) shows you the screen presented when you want to compare one volume with another.

#### *ProFile Note:*

*Be very careful when you format ProFile. It is a large storage device and may contain valuable information.*

**FORMAT A VOLUME.** This screen allows you to prepare a volume to accept programs, files, or data. All diskettes *must* be formatted before you can store data on the diskette. Every computer manufacturer uses a slightly different diskette recording scheme. The diskette manufacturers manufacture blank diskettes. Therefore, computer manufacturers provide either system or utility software that provides their recording scheme on the surface of the manufacturer's blank diskette. Figure. 8.13 shows you the screen for formatting a volume.



Figure 8.13. FORMAT A VOLUME display screen.

The surface of the recording media is divided into sections called blocks. A block of data is 512 bytes long. In DOS 3.3 terminology, this would be 2 sectors, because a sector holds 256 bytes of information. Further, there are 35 tracks of 8 blocks each where each track is marked for recording data.

Notice that you have the capability of accepting the default volume name assigned by the /PRODOS/FILER or choosing one of your own. If you accept the default, volumes will be named as follows:

/BLANKxx

In this case, the xx represent numbers that start with 00 and increment by one each time you accept a default value. The range of numbers are from 00 through 99. Then the number sequence will recycle from the beginning.

Further, you may format a volume using any legitimate slot-drive combination, including a ProFile hard disk, if you have one. So, please be careful when making selections. Remember that the ESCAPE key may always be used to restart a screen or to escape that selection.



When you enter a slot-drive combination or a volume name of your own, all you have to do is type over the default values or name presented on the screen.

You will probably have some diskettes that you previously formatted. Now you want to reformat them and store new information. When ProDOS determines that the diskette already has information stored on the diskette, you will be asked to verify that you wish to destroy all previous information. Do not answer the question lightly, because once you respond by telling the format program to destroy previous information, there is no retreating. The stored information is gone.

After a diskette is successfully formatted, you will see the message:

## FORMAT COMPLETE

Remove the formatted diskette from the disk drive and label it with the volume name, contents, and current date, and identify it as a ProDOS diskette.

### *ProFile Note:*

*If you are going to format a mass storage disk like the ProFile, you will get the message:*

## WARNING:YOU ARE ABOUT TO FORMAT A LARGE DISK

*If you made a mistake, use the ESC key to terminate the selection.  
If it is what you want, type RETURN key.*

**COPY A VOLUME.** In Chapter 1 you were introduced to making copies of the diskettes that came with ProDOS. It is always a good idea to make copies of your diskettes. Even though the data stored on a diskette is very stable and highly reliable, disasters can happen. Murphy's law again. So, take out "volume insurance" and make backup copies of your diskettes.

In the previous section, it was stated that you must format a diskette before you are able to store data and information on that diskette. Now you will be given the exception to the rule. When you copy one volume to another, the COPY A VOLUME option will first reformat the destination diskette before copying the contents of the source diskette to the destination diskette. If there is *anything* at all on the destination diskette you want to keep, PLEASE move those files to someplace else so they won't be lost.

When making a copy of a volume, you may maintain the same name or change the name on the new volume. Figure 8.14 shows you the screen for copying a volume.

The source (original) diskette in the above example is to be placed in slot 6, drive 1. The destination (new or copy) diskette is to be placed in slot 6, drive 2.

If you have a single drive system, place your source volume in drive 1 and be prepared to do a lot of diskette swapping. Fortunately, the copy program will place messages on the video screen telling you which diskette is to be placed in the disk drive.

When all of the copying has been completed, you will be given the message:

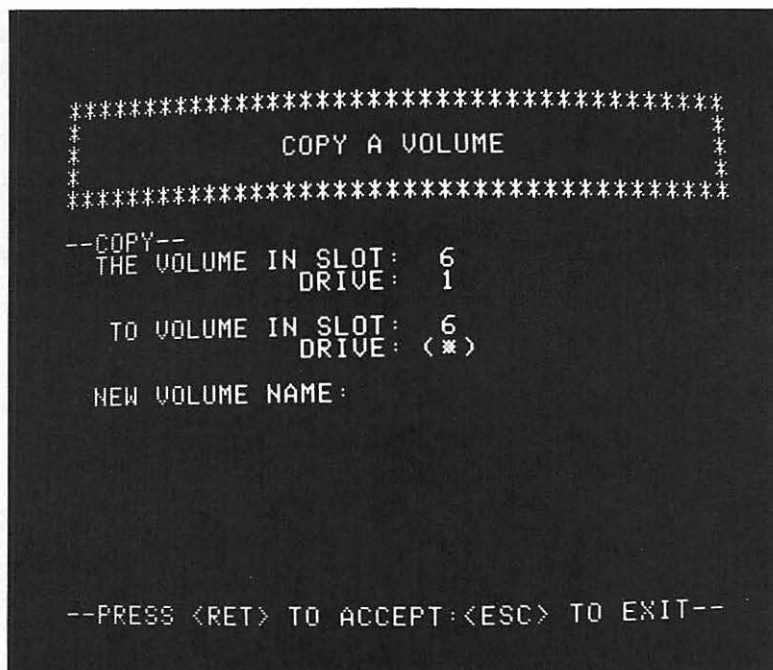


Figure 8.14. COPY A VOLUME display screen.

## COPY COMPLETE

You have once again made a successful copy of a diskette.

*Unless you have two volumes, this option is not of much value for the ProFile.*

**LIST VOLUMES.** The easiest way to determine what mass storage volumes are assigned to what expansion slots is to use this video screen command. When you select the L (LIST VOLUMES) option from the previous screen, your system will automatically test every peripheral storage device and determine the name of the volumes active. The list volumes screen is shown in Figure 8.15.

The LIST VOLUMES video screen display tells you which ProDOS volumes are active and in which expansion slot and drive they reside. If there is no volume in one of your drives, it will not be shown in the list. Further, if you have a DOS 3.3 diskette in one of your disk drives, you will see the message

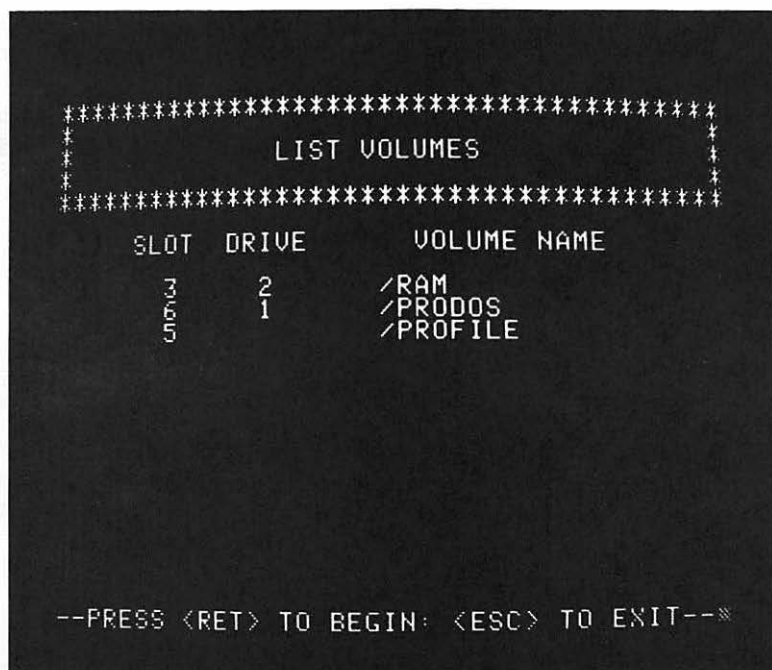


Figure 8.15. LIST VOLUMES.

## &lt;NO DIRECTORY&gt;

displayed on the video screen.

The /RAM volume is the 64K expansion memory resident on the extended 80-column text card available for the Apple IIe only. This card and its capabilities are covered in Section F.4 of Appendix F. For now, just accept the fact that it has the address of slot 3, drive 2.

*ProFile is installed in slot 5, drive 1. This is the recommended location. The ProFile is discussed in Section F.5 of Appendix F.*

**RENAME A VOLUME.** As you sit at your system formatting diskettes, it is highly probable you will think of volume names that are more interesting than /BLANK15, for example. These names may end up being whimsical, off-color or just unwanted later on. There always comes the time when the volume called /GOTO.JOHN really needs to be called /GL.MAY.84 for "General Ledger—May 1984."

The RENAME A VOLUME option allows you to make the change. This option lets you change the name of a volume without changing the contents of the diskette. The rename display is shown in Figure 8.16.

Place the volume you wish to rename into an available disk drive. Then select the slot and drive designations of the volume's location. Once you have done this you need only enter the new name for that volume. The system will do the rest.

Remember, the new name *must* have:

- the first character be alphabetic
- 15 characters or less
- no embedded blanks
- no special characters

**DETECT BAD BLOCKS.** Sometimes, it is obvious when a diskette cannot be used anymore—when it is creased in the lid of your attache case, for example, or bathed by hot coffee. Other times, volumes will go bad in much more subtle ways. They will develop bad blocks.

Bad blocks can develop in very sneaky ways from causes such as:

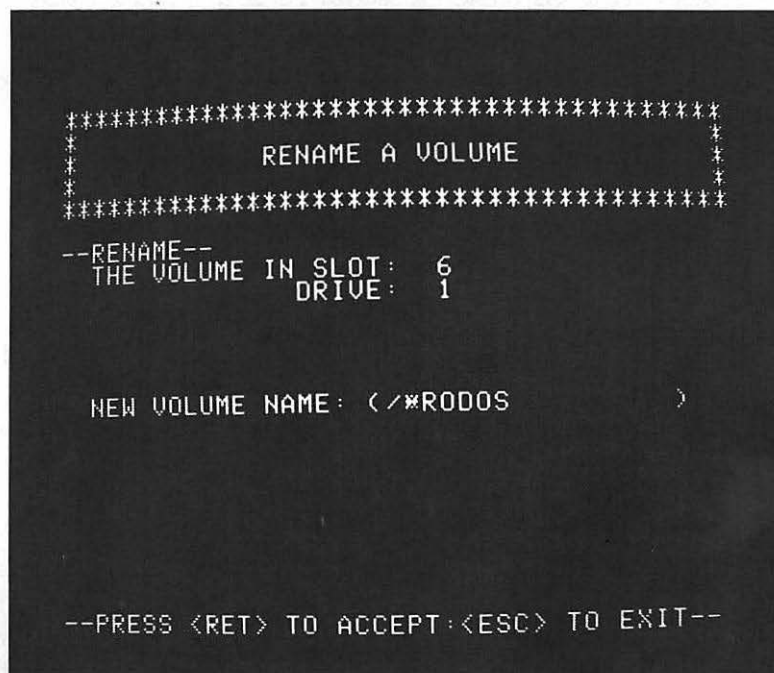


Figure 8.16. RENAME A VOLUME display screen.

- dust on the diskette
- fingerprints on the recording window
- excessive usage
- old age (like me)

If you suspect any of these things have possibly damaged a diskette volume, it is recommended that you check the volume for bad blocks. Figure 8.17 shows you the screen for doing that.

For this option all you are required to enter is the slot-drive combination that contains the diskette to be checked.

If there are no bad blocks, you will get the message:

#### 0 BAD BLOCKS

If, however, any bad blocks are discovered, these will be annotated as:

#### BAD BLOCK NUMBER

XXXX

YYYY

ZZZZ

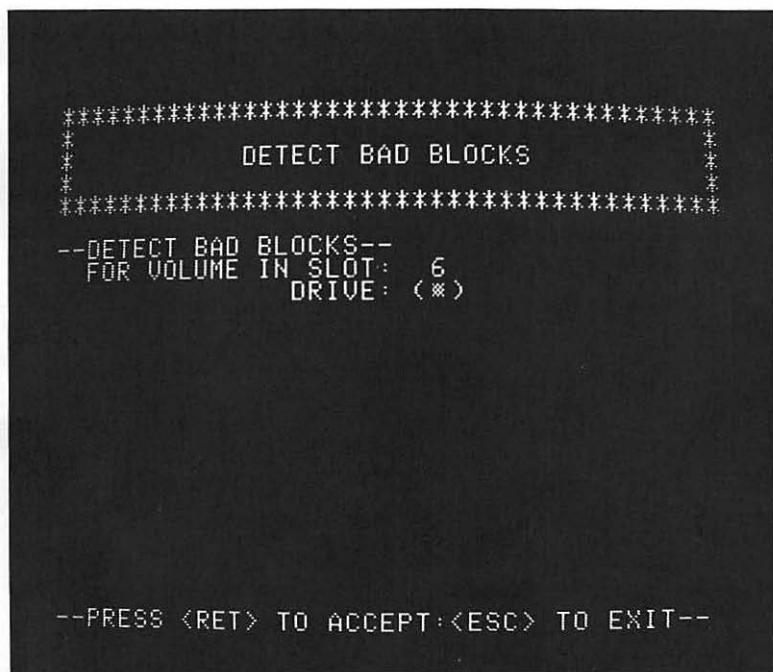


Figure 8.17. DETECT BAD BLOCKS display screen.

If you do get bad blocks detected, it is recommended that you move all of the files that are still good. Then either discard the diskette or reformat the diskette.

*Notes:*

1. You may use this option to check for bad blocks on DOS 3.3 diskettes.
2. You may also get a listing of the detected bad blocks on a printer by changing the display device. See the Configuration Defaults in Section 8.1.5.

**BLOCK ALLOCATION.** This command option allows you to find out how many blocks on a volume are used by files, how many are still available, and the total number of blocks on the volume.

This option helps a great deal when you are moving files from one volume to another. You may want to know if there is enough space left on the destination diskette to hold more files.

You should be able to get a feel for the space requirements of programs and files by using this command option.

```

*****
*                                     *
*                                BLOCK ALLOCATION                                *
*                                     *
*****
--BLOCK ALLOCATION--
  FOR VOLUME IN SLOT:  6
                    DRIVE: (1)

--PRESS <RET> TO ACCEPT:<ESC> TO EXIT--

```

Figure 8.18. BLOCK ALLOCATION screen.

**COMPARE VOLUMES.** This option, as the name says, is for the purpose of comparing two volumes for a match. This is very handy because there are times when you want to determine if your backup really is an exact copy of the original.

After you have entered the slot-drive combinations for the two diskettes you wish to compare, this subprogram does a byte-for-byte comparison of the two diskettes.

If the two volumes are really exact copies, you should get the message:

#### COMPARE COMPLETE

However, if there are any blocks that do not match, you will get the message:

#### BLOCK NUMBERS DO NOT MATCH

1  
3  
4

—PRESS <RET> TO CONTINUE: <ESC> TO EXIT

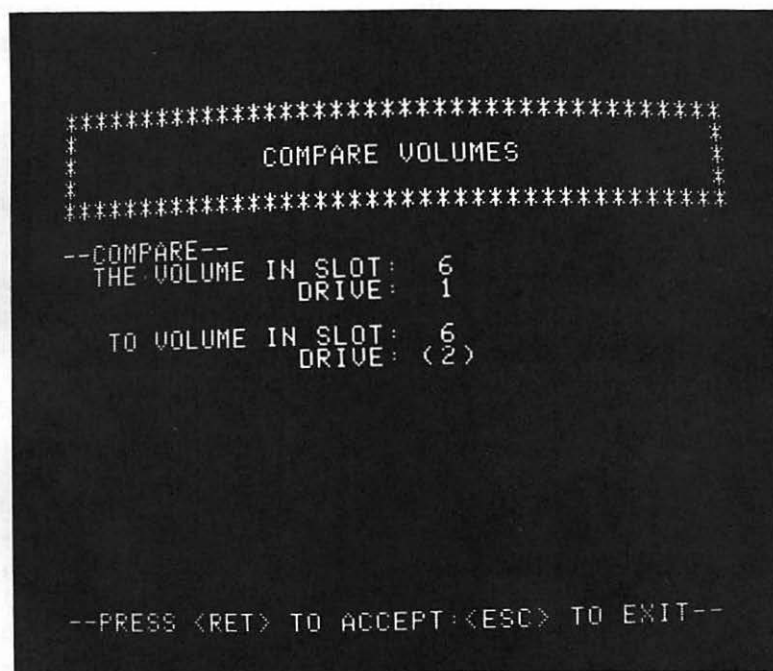


Figure 8.19. COMPARE VOLUMES display screen.

Only three mismatching blocks will be shown initially. By typing the RETURN key the remaining mismatched blocks are displayed.

*Notes:*

1. *You may use this option to compare two DOS 3.3 diskettes.*
2. *You may also get a listing of the compared diskettes on a printer by changing the display device. See the Configuration Defaults in the next section.*

Some clues to possible mismatches are:

- block 2—the names are different
- block 6—the maps for the diskettes are different

*Profile Note:*

*This option is not of much value unless you have two or more hard disks like ProFiles.*

## 8.1.5. CONFIGURATION DEFAULTS

This option allows you to customize the /PRODOS/FILER program to your particular system. You are allowed to either set your own system configuration or revert to the predefined defaults. This is shown in Figure 8.20.

When you select defaults for your system configuration, the selected values will be written to the diskette so the program must be in drive 1 of the boot drive throughout the operation.

After you have selected the S option, you will be given a secondary screen shown in the next section.

**SELECT DEFAULTS.** The SELECT DEFAULTS option will present the following screen. If you do not remember the slot assignments for your disk drives and printer then use the ESC key to return to the main menu and ask the program to display the slot assignments. See Figure 8.21. Then return to here for any reassignments you wish to make.

If you use the RETURN key on any of the values presented, then you are accepting the default value. At any time you make a mistake you may use the ESC key to return you to the top of the selections and reenter the selections.

The output device is where the FILER program will display information. If you select the printer, then all subsequent output will be sent to your printer. Make sure that your printer is turned on and is in the ON-LINE or SELECTED condition. Of course, paper in the printer will also help.

The other screen possible is to be able to restore defaults. That screen is shown in the next section.



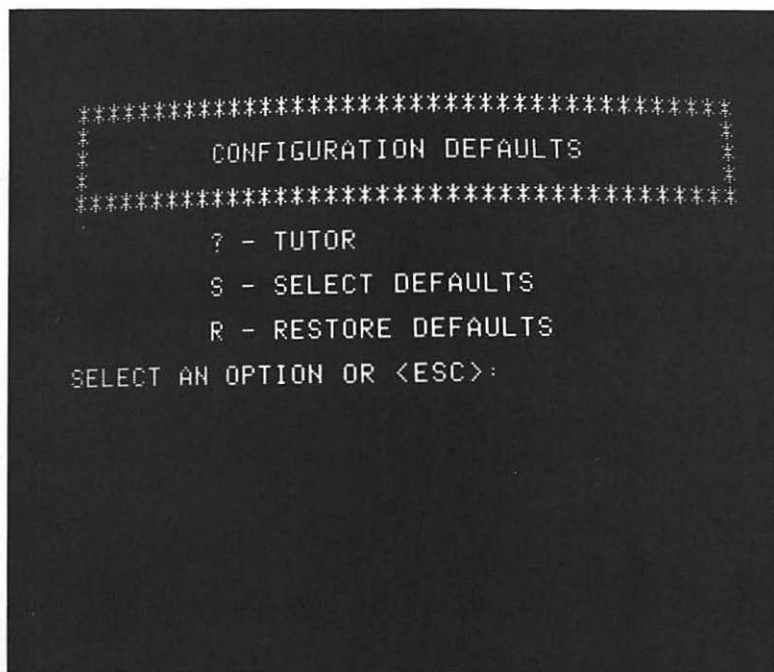


Figure 8.20. CONFIGURATION DEFAULTS screen.

**RESTORE DEFAULTS.** By selecting the RESTORE DEFAULTS option, the FILER program will restore all options to the standard or defaulted values. The screen presented is shown in Figure 8.22.

Notice that this screen shows that the disk drives are installed in slot 6 and the normal output is to be to the video display. If you wish to accept the defaults as displayed, you need only press the RETURN key.

If you decide to keep the customized values then press the ESC key and go back to where you started.

### 8.1.6. QUIT

This option is for the purpose of leaving the /PRODOS/FILER program and being able to execute any other program or system of programs.

Notice that the video screen shows that the system program, /BASIC.SYSTEM, will be executed if you accept that pathname. If you do not want to accept, simply type in the pathname of the program you want to execute.

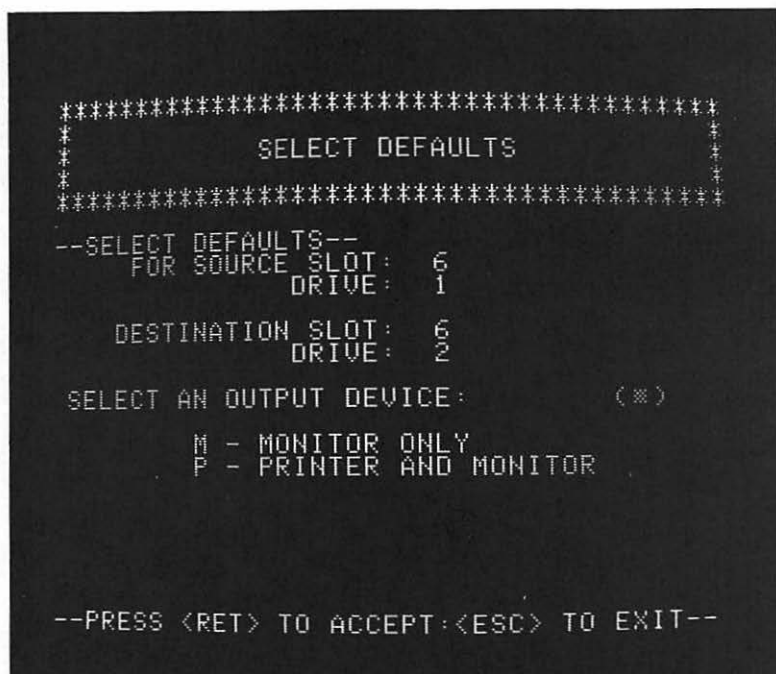


Figure 8.21. SELECT DEFAULTS screen.

## 8.2. USING THE CONVERT PROGRAM

The /CONVERT program has been included in ProDOS for the purpose of being able to convert DOS 3.3 programs and files to the new ProDOS environment. This program performs a similar function to the MUFFIN program provided when the transition was made from DOS 3.2.1 to DOS 3.3.

Because of the formatting differences between DOS 3.3 and ProDOS, you will have to convert all DOS 3.3 diskettes to ProDOS.

There are a few cautions, however. The CONVERT program will not transfer DOS 3.3 or ProDOS random-access files. You will have to find some other way to transfer these files.

### 8.2.1. ProDOS CONVERT menu

The main menu gives you the ability to choose any of the capabilities of the CONVERT program. If you do not understand how to use the program then select the TUTOR option first by typing a question mark on the keyboard.

```

*****
*                                     *
*               RESTORE DEFAULTS      *
*                                     *
*****

--RESTORE DEFAULTS--
  FOR SOURCE SLOT:  6
    DRIVE:         1

  DESTINATION SLOT:  6
    DRIVE:         2

SELECT AN OUTPUT DEVICE:             M

      M - MONITOR ONLY
      P - PRINTER AND MONITOR

--PRESS <RET> TO ACCEPT:<ESC> TO EXIT--

```

Figure 8.22. RESTORE DEFAULTS screen.

It is recommended that you set the date first if there is no date showing. This will give you the created date on your ProDOS diskette when you transfer files.

### 8.2.2. Reverse transfer direction

This option allows you to change the direction of the transfer. When you select this option, the screen will only change in the direction line. This direction display actually wraps around in this area.

### 8.2.3. Change DOS 3.3 slot and drive

This option lets you adjust the slot-drive combination to suit your particular system. When selecting this option, you may either enter new information, enter partial information or accept defaults by simply typing the RETURN key.

#### *Profile Note:*

*When transferring files from ProFile to DOS 3.3, change the slot-drive combination first.*

```

*****
*                                     *
*                               QUIT   *
*                                     *
* **PREFIX: /PRODOS/*****          *
*
--QUIT AND LOAD--
  PATHNAME: (BASIC.SYSTEM*)

--ENTER PATHNAME AND PRESS <RET>--

```

Figure 8.23. QUIT screen.

```

                        CONVERT Menu
Direction: DOS 3.3 S6/D2 ---> ProDOS
Date: 30-JAN-84
Prefix: /PRODOS/

-----

R - Reverse Direction of Transfer
C - Change DOS 3.3 Slot and Drive
D - Set ProDOS Date
P - Set ProDOS Prefix
T - Transfer (or List) Files

-----

Enter Command: ?   ? - Tutor,  Q - Quit

```

Figure 8.24. CONVERT main menu.

### 8.2.4. Set ProDOS prefix

This option allows you to change the ProDOS prefix for the purpose of converting programs and files from many different diskettes to many different diskettes. Make sure that you have entered the correct prefix before attempting to transfer any files.

### 8.2.5. Set ProDOS date

Since there is a place for the modification and creation date when you place files on a diskette, you need to have a way to date-stamp your files.

### 8.2.6. Transfer or list files

This option is probably the one that you will use the most. This option allows you to list the files on the DOS 3.3 diskette, and then from that presented list move the files to the ProDOS diskette.

## 8.3. FILER AND CONVERT ERROR MESSAGES

Appendix D gives you a list of the possible error messages provided by both /PRODOS/FILER and /CONVERT.

## SUMMARY

This chapter discussed the two major utility software programs: /PRODOS/FILER and /CONVERT. You were shown how each option worked and what each accomplished.

The /PRODOS/FILER program includes the following subprograms:

### ?-TUTOR

#### F-FILE COMMANDS

- ? -TUTOR
- L-LIST PRODOS DIRECTORY
- C-COPY FILES
- D-DELETE FILES
- K-COMPARE FILES
- A-ALTER WRITE-PROTECTION
- R-RENAME FILES
- M-MAKE DIRECTORY
- P-SET PREFIX

#### V-VOLUME COMMANDS

- ? -TUTOR
- F-FORMAT A VOLUME
- C-COPY A VOLUME
- L-LIST VOLUMES

R-RENAME A VOLUME  
D-DETECT BAD BLOCKS  
B-BLOCK ALLOCATION  
K-COMPARE VOLUMES  
D-CONFIGURATION DEFAULTS  
? -TUTOR  
S -SELECT DEFAULTS  
R-RESTORE DEFAULTS  
Q-QUIT

The /CONVERT program includes the following subprograms:

R Reverse Direction of Transfer  
C Change DOS 3.3 Slot and Drive  
D Set ProDOS Date  
P Set ProDOS Prefix  
T Transfer (or List) Files  
  
? Tutor  
Q Quit

The major emphasis in this chapter was the description of how to operate the /PRODOS/FILER and /CONVERT programs.

## QUESTIONS

1. How do you get the /PRODOS/FILER and /CONVERT programs to execute?
2. Describe the functions of /PRODOS/FILER in detail.
3. Describe each of the volume commands used in the /PRODOS/FILER program.
4. Describe each of the file commands used in the /PRODOS/FILER program.
5. Describe the cautions and considerations when using the ProFile.
6. Discuss how to customize the /PRODOS/FILER program to your own system.
7. What options do you have available when you quit the /PRODOS/FILER program?

# 9. THE MACHINE LANGUAGE INTERFACE

*If it (automation) keeps up, man will  
atrophy all his limbs but the  
push-button finger.*

*Frank Lloyd Wright, 1955*

## 9.0. OVERVIEW

This chapter assumes that you have had some experience with 6502 assembly language and the internal structure of the Apple II family of computers. The intent of this book is to explain ProDOS in enough depth to allow you to take advantage of this new operating system. It is recommended that you refer to any of the numerous excellent books on 6502 assembly language to give you additional information. Further, you may need to study the ProDOS technical manual, if you need more in-depth information.

ProDOS, in addition to being an operating system, also handles interrupts, provides memory management, and time-date stamps files from a clock/calendar card, if you have one installed in your computer.

There is a new mechanism in ProDOS called the Machine Language Interface (MLI). This mechanism allows you to make calls to the operating system, validates them, and issues

operating system commands. Calls to the MLI give you control over certain hardware. MLI calls may be categorized into housekeeping, memory, and interrupt calls. This chapter will discuss these mechanisms.

In this chapter and in others, the term *system program* has been used. This term as used here and by Apple Computer, Inc. in their ProDOS manuals may be a bit confusing, especially if you are familiar with system programs on large computer systems. A system program on large computer systems is neither an applications program (for instance general ledger or financial planner) nor an operating system (for example OS 360/370 or OS/VMS). It normally provides a means of making operating system calls from an application program (for instance a sorting module or file management module).

Under ProDOS, a system program refers to any program written in assembly language that makes calls to the Machine Language Interface (MLI), and follows those protocols or conventions. System programs may be identified by their file type or by their name, of the form ---.SYSTEM. In general, it is the structure of a program, not its function, that makes a program a ProDOS system program.

## 9.1. MEMORY USAGE

This section discusses the way the Machine Language Interface uses memory and various memory areas within ProDOS. ProDOS treats memory in the same way the 6502 microprocessor treats memory: as a sequence of 256-byte pages. These pages are numbered \$00=0 through \$FF=255 for every 64K=65536 memory space.

Section 9.1.1 gives the booting and loading sequence required to install ProDOS into memory and activate the operating system.

Section 9.1.2 gives a number of memory locations and their translations. ProDOS maintains a system bit map that keeps track of the usage status of every page of memory. More will be said about this later in this section.

### 9.1.1. ProDOS loading sequence

When you first boot up your system with a startup diskette, a reasonably complex loading procedure is set into motion. This loading procedure is set into motion any of a number of ways. These are:

1. Applying power to a turned off machine with a ProDOS diskette installed in your boot drive.
2. Issuing a PR# or IN# command from the Applesoft II BASIC immediate mode.
3. Issuing a 6-(CTRL)-P from the monitor prompt.

Any of the above causes a transfer of control to the onboard ROM located on the disk controller card. As this transfer takes place, the following procedure will begin:

1. The disk drive ROM program is executed, which causes blocks 0 and 1 from the diskette to be loaded into memory at \$800 = 2048, and then that program is executed.



2. The loader program attempts to find the file named PRODOS with the file type \$FF = 255 in the volume directory of the startup diskette. The \$FF file type is a ProDOS system file. The PRODOS file is loaded into memory at location \$2000 = 8192. After the program is loaded, then this program is executed.
3. Then the MLI determines your computer's memory size and relocates itself to its final location. The program then determines the devices installed and their slot locations. Finally, the system global page in memory is set up.
4. The last procedure accomplished is a search made of the volume directory for the first file with the name ---.SYSTEM and a type \$FF = 255 file. Once this is loaded and relocated it is executed.

If either the PRODOS or ---.SYSTEM program files are missing, your system will report to you that it is unable to load ProDOS. The memory map for ProDOS and those for other memory areas are shown in Appendix B.

Once you have ProDOS installed, the search order for files is:

- slot 6, drive 1
- slot 6, drive 2
- slot 7
- slot 5, drive 1
- slot 5, drive 2

### 9.1.2. Memory maps

The following memory sections and their translations show you a number of places within ProDOS that you may find interesting or helpful. These, along with numerous subroutines, should help you understand the makeup of ProDOS.

The ProDOS MLI uses locations \$40 through \$4E on page zero during calls. These location values are saved before an MLI call and then restored before exiting that call. The page zero locations \$3A through \$3F are used for disk-driver routines. These are not restored after usage.

A series of constants held in memory are shown below.

```

B898- . . . . . AE AF B0 . . . . . / 0
B8A0- B1 B2 B3 B4 B5 B6 B7 B8 1 2 3 4 5 6 7 8
B8A8- B9 . . . . . 9 . . . . .

```

The memory locations \$B8CD through \$B93A contain the ProDOS command table. There are some very interesting things to be noticed from this table. First, the DOS 3.3 command table set the high bit on the last character of each particular command. In ProDOS, that feature has been eliminated. Second, each command was listed separately. In ProDOS, the commands have been run together, bunched up, and have no separators. This means that the command table takes less space and implies a very different means of parsing for legal commands.

```

B8C8- . . . . . 42 53 41 . . . . . BSA
B8D0- 56 45 52 49 46 59 42 4C V E R I F Y B L
B8D8- 4F 41 44 45 4C 45 54 45 O A D E L E T E
B8E0- 43 41 54 41 4C 4F 47 4F C A T A L O G O
B8E8- 50 45 4E 57 52 49 54 45 P E N W R I T E
B8F0- 58 45 43 52 45 41 54 45 X E C R E A T E
B8F8- 46 52 45 53 54 4F 52 45 F R E S T O R E
B900- 4E 41 4D 45 42 52 55 4E N A M E B R U N
B908- 4C 4F 43 4B 43 48 41 49 L O C K C H A I
B910- 4E 23 46 4C 55 53 48 52 N # F L U S H R
B918- 45 41 44 50 4F 53 49 54 E A D P O S I T
B920- 49 4F 4E 4F 4D 4F 4E 50 I O N O M O N P
B928- 52 23 50 52 45 46 49 58 R # P R E F I X
B930- 43 4C 4F 53 45 41 50 50 C L O S E A P P
B938- 45 4E 44 . . . . . E N D . . . . .

```

The next memory locations show the ProDOS command modifier translations.

```

B9B8-. 41 42 45 4C 53 44 46 . A B E L S D F
B9C0- 52 56 . . . . . R V . . . . .

```

The following memory locations show the translations for the various file type abbreviations as they are displayed when you use the CAT or CATALOG commands.

```

B9E0-. . . . . C1 C4 C2 . . . . . A D B
B9E8- C1 D3 D0 C1 D7 D0 D0 C1 A S P A W P P A
B9F0- D3 D4 D8 D4 C2 C9 CE C4 S T X T B I N D
B9F8- C9 D2 C3 CD C4 C9 CE D4 I R C M D I N T
BA00- C9 D6 D2 C2 C1 D3 D6 C1 I V R B A S V A
BA08- D2 D2 C5 CC D3 D9 D3 . R R E L S Y S .

```

The following memory locations show a series of month names that are used to display dates when ProDOS needs to display the date.

```

BA0F-. . . . . C A . . . . . J
BA10- C1 CE C6 C5 C2 CD C1 D2 A N F E B M A R
BA18- C1 D0 D2 CD C1 D9 C A D 5 A P R M A Y J U
BA20- C E C A D 5 C C C1 D5 C7 D3 N J U L A U G S
BA28- C5 D0 CF C3 D4 CE CF D6 E P O C T N O V
BA30- C4 C5 C3 BC CE CF A0 C4 D E C < N O D
BA38- C1 D4 C5 BE 28 AB 40 41 A T E >

```

```

BA3C- . . . . 28 AB 40 41 . . . . ( + @ A
BA40- 42 43 44 45 46 47 48 49 B C D E F G H I
BA48- 4B 4C 4D 4E 50 53 56 . K L M N P S V .

```

The next few memory locations show a prompting message that may be placed upon a video screen when for any reason you are left in the monitor.

```

BB98- . . . . D0 CC C5 C1 D3 . . . P L E A S
BBA0- C5 A0 D0 D2 C5 D3 D3 A0 E P R E S S
BBA8- D3 D0 C1 C3 C5 A0 C2 C1 S P A C E B A
BBB0- D2 A0 00 00 00 00 00 00 R . . . . .
BCC0- 30 2E 42 45 30 2E 44 49 O . B E O . D I
BCC8- 53 4B 2F 53 54 41 52 54 S K / S T A R T
BCD0- 55 50 . . . . . U P . . . . .

```

The \$BE54 and \$BE55 memory locations are used to handle the parsing of any command string parameters. The various bit positions within these bytes have distinct hexadecimal values. Each of these values has a distinct meaning. These meanings are shown below.

The memory locations \$BD00 through \$BDFF have uses that are unknown at this time.

The memory locations \$BE54 and \$BE55 contain the modifier values allowed for a specific ProDOS command.

```

BE54 = $80  Need a prefix, pathname is optional
          40  No parameters needed to be processed
          20  Deferred mode command only
          10  Filename is optional
          08  CREATE command allowed
          04  File type optional
          02  Second filename required for RENAME command
          01  A filename is expected

```

```

BE55 = $80  Address allowed
          40  Byte allowed
          20  End address allowed
          10  Length value allowed
          08  @ line number allowed
          04  Slot and drive numbers allowed
          02  Field
          01  Record
          00  Volume number ignored

```

The following memory locations are a series of bytes with the high bit set to 1, and translate to the copyright message sometimes displayed by ProDOS on a video screen.

```
BEE0- C3 CF D0 D9 D2 C9 C7 C8  C O P Y R I G H
BEE8- D4 A0 C1 D0 D0 CC C5 AC  T A P P L E ,
BEF0- A0 B1 B9 B8 B3 . . . 1 9 8 3
```

The \$BF = 191 page of memory contains the system's global variables. This includes the addresses \$BF00 through \$BFFF. This section of memory is inviolate. Because of this, this page serves as the communication link between system programs and the operating system. The MLI places all information that is helpful or useful to a system program into these memory locations.

```
BF00- 4C B7 BF  JMP  $BFB7 ;MLI call entry point
BF03- 4C BD EE  JMP  $EEBD ;Spare. Reserved for future use.
BF06- 4C 42 F1  JMP  $F142 ;Clock/Calendar routine, user installed
BF09- 4C DA D1  JMP  $D1DA ;Error reporting vector.
BF0C- 4C E6 D1  JMP  $D1E6 ;System failure vector.
BF0F- 00                      ;Error code. 0 = No error.
```

In the next memory locations are a series of two-byte addresses starting at \$BF10. Each of these addresses contains information about specific devices. The range of memory from BF10 through BF2F contains information defined as follows:

BF10: Slot zero reserved	BF20: Slot zero reserved
BF12: Slot 1, drive 1	BF22: Slot 1, drive 2
BF14: Slot 2, drive 1	BF24: Slot 2, drive 2
BF16: Slot 3, drive 1	BF26: Slot 3, drive 2 = / RAM, reserved
BF18: Slot 4, drive 1	BF28: Slot 4, drive 2
BF1A: Slot 5, drive 1	BF2A: Slot 5, drive 2
BF1C: Slot 6, drive 1	BF2C: Slot 6, drive 2
BF1E: Slot 7, drive 1	BF2E: Slot 7, drive 2

If one of these addresses carries the value D0A2, that slot is currently empty and does not contain an interface card. Listed below are possible entries for a system.

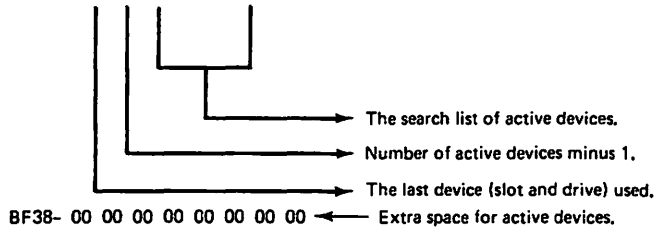
```
BF10- A2 D0 A2 D0 A2 D0 A2 D0
BF18- A2 D0 EA C5 00 FB A2 D0
BF20- A2 D0 A2 D0 A2 D0 00 FF
BF28- A2 D0 A2 D0 00 F8 A2 D0
```

### Notes

1. Slot 5, drive 1 (BF1A-BF1B) contains a disk drive. In this case a ProFile hard disk.
2. Slot 6, drive 1 (BF1C-BF1D) contains a Disk II drive.

3. Slot 3, drive 2 (BF26-BF27) contains /RAM as a memory disk drive.
4. Slot 6, drive 2 (BF2C-BF2D) contains a Disk II drive.

BF30- 60 03 54 E0 60 BF 00 00



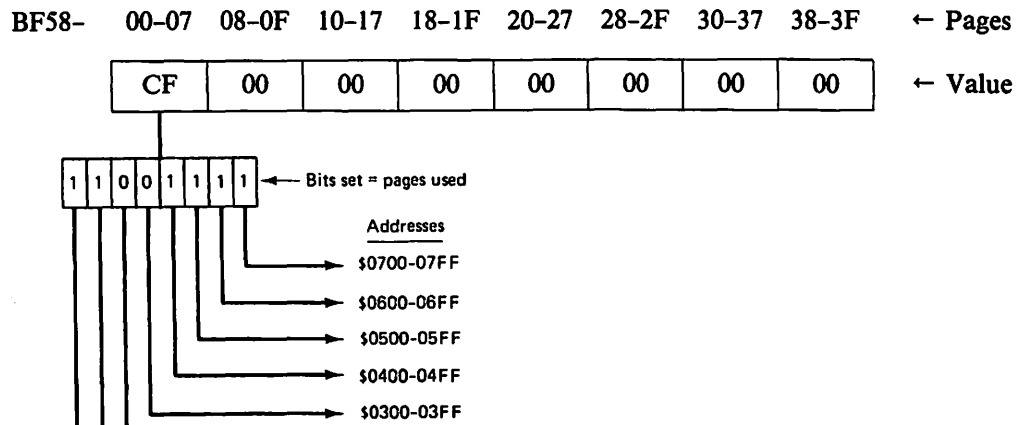
The following memory locations are for the purpose of putting a message on the video screen.

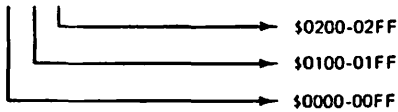
BF40- 43 4F 50 52 2E 20 41 50 C O P R . A P  
 BF48- 50 4C 45 2C 31 39 38 33 P L E , 1 9 8 3

The purpose of the following eight memory locations is not known at this time.

BF50- 8D 8B C0 4C D8 FF 00 00

The following addresses are the memory map of the lower 48K. Each bit that is set represents one page of memory (256 bytes) that is currently in use. Those pages that are in use (protected) are marked with a 1 (set). Those that are not in use (unprotected) are marked with a 0 (reset). ProDOS will not allow you to write file buffer information into protected areas. This area is known as the system bit map. There are 24 bytes set aside for the system bit map. Each byte represents a 2K block of memory. Therefore, 8 bytes represents 16K of memory. The 24-byte system bit map represents 48K of memory. How each 256 bytes of memory is represented is shown below.





The next 16K of memory is shown as being unused.

BF60- 40-47 48-4F 50-57 58-5F 60-67 68-6F 70-77 78-7F ← Pages

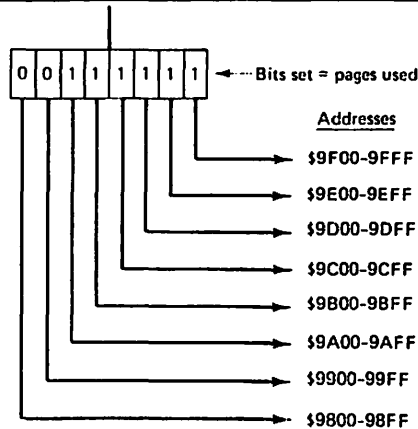
00	00	00	00	00	00	00	00
----	----	----	----	----	----	----	----

← Value

BF68- 80-87 88-8F 90-97 98-9F A0-A7 A8-AF B0-B7 B8-BF ← Pages

00	00	00	3F	FF	FF	FF	C3
----	----	----	----	----	----	----	----

← Value



The following eight memory locations are used to hold the addresses of the allowable file buffer starting addresses. These are for the open files only. They should not be changed when activated. None are active at this time.

BF70- 00 00 00 00 00 00 00 00

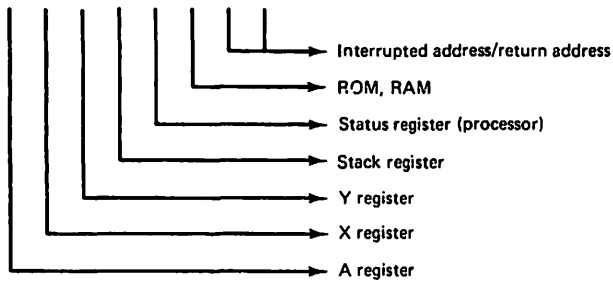
BF78- 00 00 00 00 00 00 00 00

The next eight memory locations are for the purpose of holding the addresses of the allowable four interrupt vectors. Again, these should not be changed.

BF80- 00 00 00 00 00 00 00 00

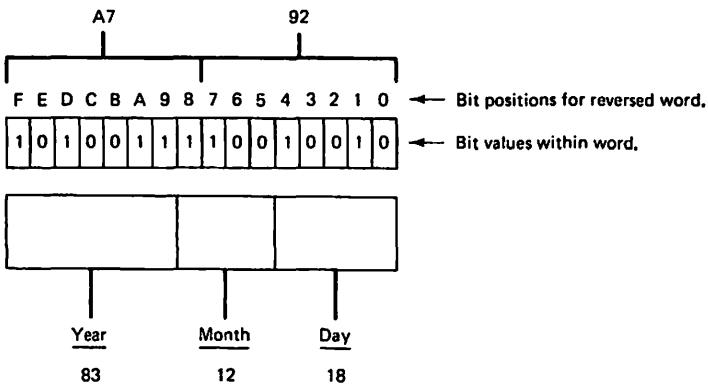
The memory addresses BF88 through BF8F are for the purpose of holding the values and status of various 6502 processor and memory areas.

BF88- 00 00 00 00 00 00 00 00

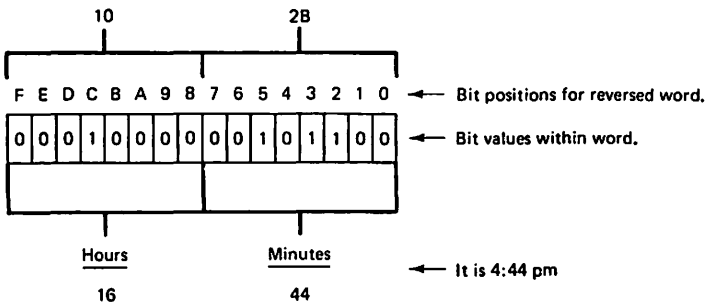


The bytes BF90 through BF91 hold the current date and BF92 through BF93 hold the current time, provided you have a clock/calendar card installed in your system that is supported by ProDOS. Note that it is necessary to reverse the bytes for the purpose of figuring the current date and time. This is done below.

BF90- 92 A7 — These two bytes hold the date.



BF92- 2B 10 — These two bytes hold the time.



Both the date and time bytes are used when writing call to the MLI discussed in the next section.

**BF94-** File level number. Used during OPEN, FLUSH, and CLOSE. When a file is opened it is assigned a level from 0 through 4 depending upon the value in this location.

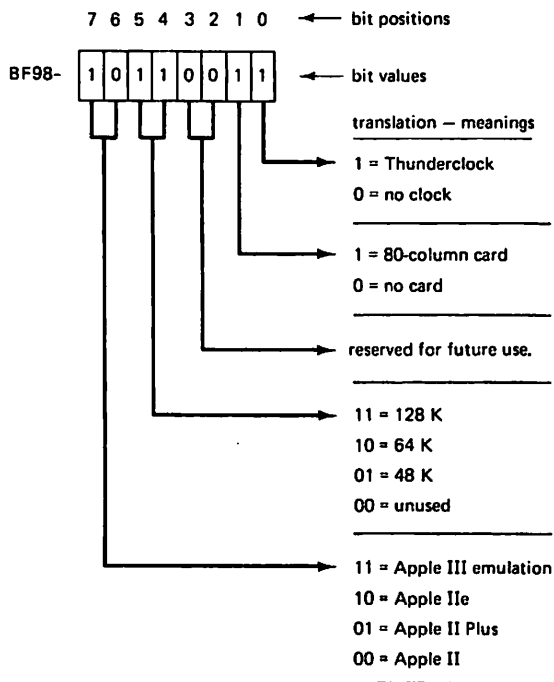
**BF95-** Backup byte.

The next two bytes are reserved for future use.

**BF96-**

**BF97-**

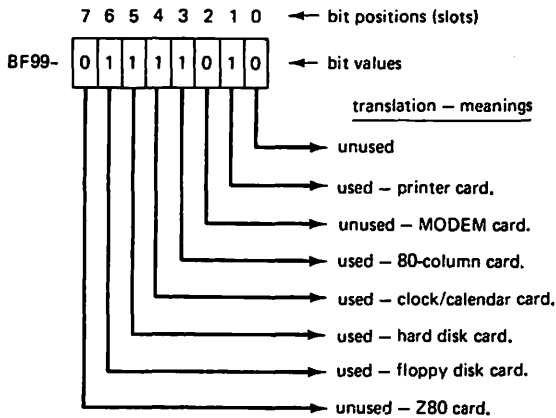
The location \$BF98 is the machine identification byte. Each of the bits within the byte have special meanings. For example:



From the bit pattern above, the computer is an Apple IIe with 128K of memory that also has an 80-column card and a Thunderclock card. From this byte you have a great deal of information.

The next byte will tell you which slots are occupied with ROM installed. For example:





For the example, slots 2 and 7 are not used. To the right of the diagram are listed the normal or default cards that may be installed in the individual slots. Most manufacturers design their products to be slot independent. However, over the past 6 years, certain slots have taken on specific peripherals. Those are the ones listed.

- BF9A-** This byte is for the purpose of storing whether a prefix is active or not. A 0 means that no prefix is active.
- BF9B-** This byte shows whether the MLI is active or not. If bit 7 is turned on (\$80) then the MLI is active.
- BF9C-** These two bytes carry the return address of the last MLI call.
- BF9D-**
- BF9E-** This byte carries the X register value on entry to the MLI.
- BF9F-** This byte carries the Y register value on entry to the MLI.

The memory areas **\$BFA0** through **\$BFFB** contain a series of routines for the language card bank-switching. These addresses are subject to change.

- BFFC- 00** The earliest version of the MLI with which your system program will work. In this case zero (00).
- BFFD- 00** The version number of your system program. In this case also a zero (00).

BFFE- 00 This byte carries the minimum compatibility version.

BFFF- 00 This byte carries the latest version number of ProDOS.

## 9.2. ISSUING CALLS

The MLI provides a very handy interface between the machine-language programmer and files stored on a diskette. The MLI is totally independent of the BASIC.SYSTEM program. The MLI consists of:

- The command dispatcher
- The block file manager
- Disk driver routines
- The interrupt handler

In order to use the MLI there are a couple of very simple requirements that need to be followed. The following machine-language routine is actually about all that is required.

```
JSR MLI      ;Call command dispatcher located at address $BF00
DB CNUM      ;This byte defines which call is being made
DW PLIST     ;A two-byte address pointer to the parameter list
BNE ERROR    ;Branch to error routine if accumulator is nonzero
```

The JSR is a Jump to SubRoutine located at the MLI address. This address is \$BF00. The DB means Define Byte. The value placed there is the call number (CNUM). The DW means Define Word. This is a 2-byte field that carries the address pointer to the parameter list allowed for this call. PLIST means Parameter LIST. The last line of code is BNE, Branch if accumulator is Not Equal to zero. This implies that the accumulator of the 6502 processor *must* be set to a zero value provided the call terminates successfully. Another instruction that may be used is the BCS (Branch if Carry is Set). This implies that you must set the carry flag to zero (CLear the Carry, CLC) if the call terminates successfully, and make sure the accumulator is zero. If a call is to terminate in an error then you should SEC (SEt the Carry) and load the accumulator with the error code value before returning from the subroutine.

By the way, a call to the MLI will return to the JSR + 3 bytes. This means that you will be returned to the BNE instruction. The only 6502 processor flags that are affected are the Z-flag and the C-flag. The Z-flag is set to 1 if the accumulator is zero. You have to set the C-flag if an error is returned in the accumulator. Further, you should make arrangements in your code to preserve the X and Y register and the SP (stack pointer).

The parameter list defined by the PLIST pointer will contain information to be used by the MLI call. There are three types of elements used in a parameter list. These are values, results, and pointers.

A value is a byte or bytes that carry quantities used by the MLI routine or the Block File Manager (BFM).

A result is a byte or bytes in the parameter list where the BFM will place values. From these values, programs can get information about the results of performing the MLI call.

A pointer is normally a 2-byte memory address that will indicate a location in memory where data, code, or space is available for the storage of information.

Finally, the first element in the parameter list is the parameter count. This is a 1-byte value that tells you the number of parameters in the list to be used by the MLI call. This value does not include the parameter count byte.

MLI calls are divided into three distinct groups. Each of these groups is discussed in one of the next three sections.

Fortunately, there is a program called EXERCISER supplied with ProDOS. This program will help you learn how to use the MLI. It allows you to execute MLI calls from a menu. You will use this exerciser program just as if you were writing a call of your own. You specify the call by its command number, the parameter list, and then execute the MLI call by typing a carriage RETURN.

When you run this program you use the command:

]— /PRODOS/ EXERCISER

The screen you are presented is shown in Figure 9.1.

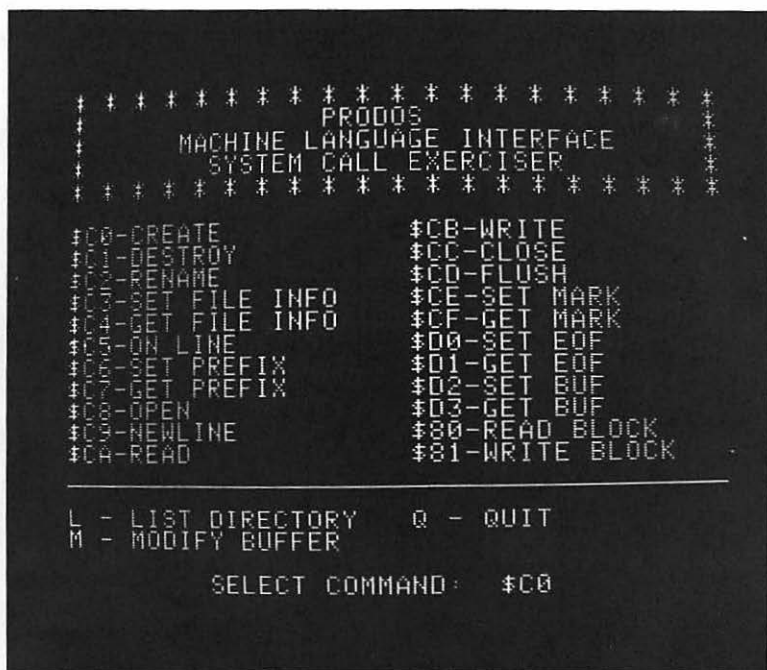


Figure 9.1. EXERCISER main menu.

At this point you can exercise each of the commands in the list presented and gain confidence that writing calls to the MLI is not difficult.

The next sections will describe the above calls. After a description of the call there is a screen presentation of the requirements for that call as presented by the EXERCISER. By using the EXERCISER along with the explanations of the MLI calls, you will quickly become familiar with this capability.

### 9.2.1. Housekeeping calls

This section discusses calls akin to the housekeeping commands of ProDOS. In most cases, these commands involve the status of a file or volume but not necessarily the contents of the file.

#### CREATE: (\$CO)

This command creates either a standard file or a directory file. A single block of disk space is allocated when using this call.

You must use this call to create every file except the volume directory. In doing that there are two distinct types of file storage. These are:

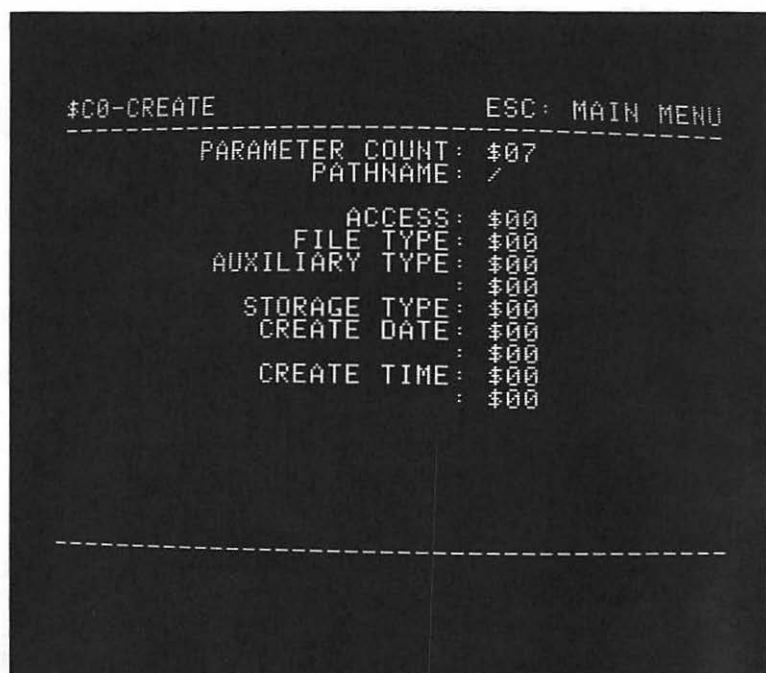


Figure 9.2. CREATE call.

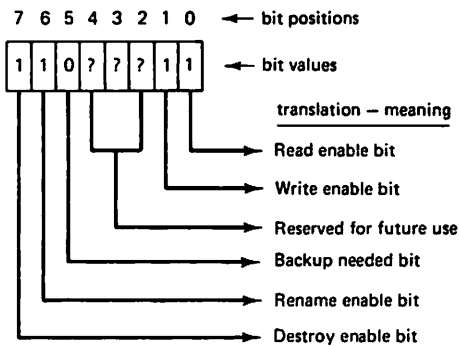
- Tree structure      type = \$01
- Linked list        type = \$0D

The tree structure storage type of file is used for all standard files, such as text and program files. The linked list storage type is used for all directory files.

The **PARAMETER COUNT** entry for this call is 7.

For the **PATHNAME** enter the complete pathname for the file or subdirectory. When using this call, enter a 2-byte address that points to an ASCII string that contains the pathname. This string contains a count byte plus a string of up to 64 characters. If you enter a complete pathname, make sure it starts with a slash, “/”, character. If not, the string is treated as a partial pathname and the current prefix is attached to the front of the file name.

The **ACCESS** byte defines how the file will be accessible. The bit meanings are as shown below:



A value of 1 in a bit position indicates that access is enabled. Therefore, a zero means that access is disabled. In general, it is recommended that you use \$C3 for the access byte.

The **FILE TYPE** and **AUXILIARY TYPE** entries may be anything you desire, however, it is recommended that you use the standard types. See Section 3.3 for file type designation numbers. The **AUXILIARY TYPE** field is used for other purposes. **BASIC.SYSTEM** uses it to store a text file record size or, in the case of a binary file, the load address.

The **STORAGE TYPE** byte is used to store the storage type for the file. This may be either a \$01 or a \$0D as in the **FILE TYPE** byte.

The **CREATE DATE** and **CREATE TIME** 2-byte word is to contain the date and time in the format shown in memory location \$BF90 through \$BF93 in Section 10.1.

## DESTROY: (\$C1)

This call will remove a standard file or directory from the diskette. In order to destroy a directory is must be empty of all files. The only exception is a volume directory, which may only be destroyed by reformatting the diskette medium. When you destroy a file, the file's name is removed from the directory and the file space is released.

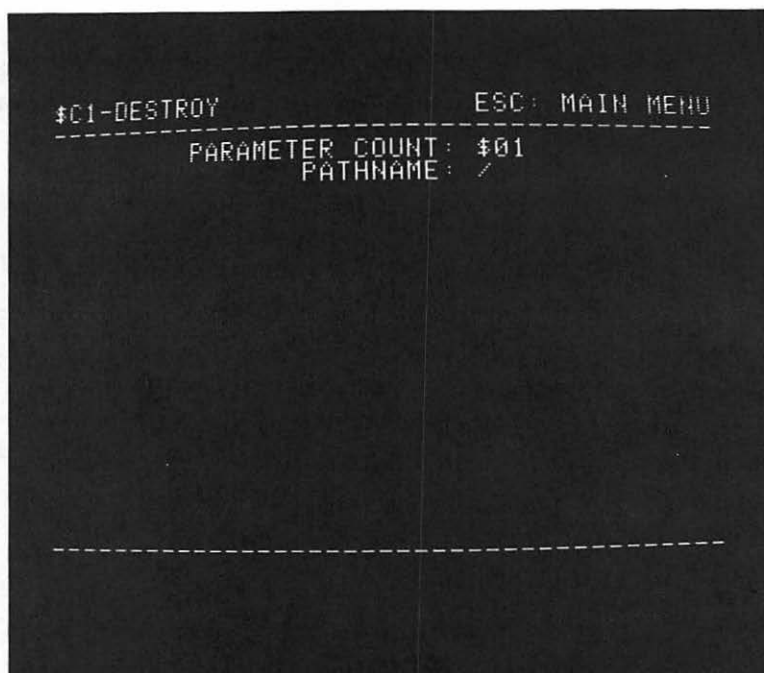


Figure 9.3. DESTROY call.

The parameter count for this call is 1.

For the EXERCISER program you enter the pathname for the file you wish to destroy. For a call that you write, enter an address pointer to the ASCII string that contains the pathname. The string contains a count byte followed by the string itself. If the pathname begins with a slash, “/”, character, it is treated as a complete pathname of 64 characters or less. If not, it is treated as a partial pathname and the prefix is placed in front of the file name.

#### RENAME: (\$C2)

This call changes the name of a file. The new name must be in the same directory and be unique for that directory.

The parameter count for this call is 2.

For the EXERCISER program you enter the pathname for the file you wish to destroy. For a call that you write, enter an address pointer to the ASCII string that contains the pathname. The string contains a count byte followed by the string itself. If the pathname begins with a slash, “/”, character it is treated as a complete pathname of 64 characters or less. If not, it is treated as a partial pathname and the prefix is placed in front of the file name.

The second pathname must follow the same rules as the first pathname. For the EXERCISER program use the pathname. For the call you write, use the address of the pathname string.

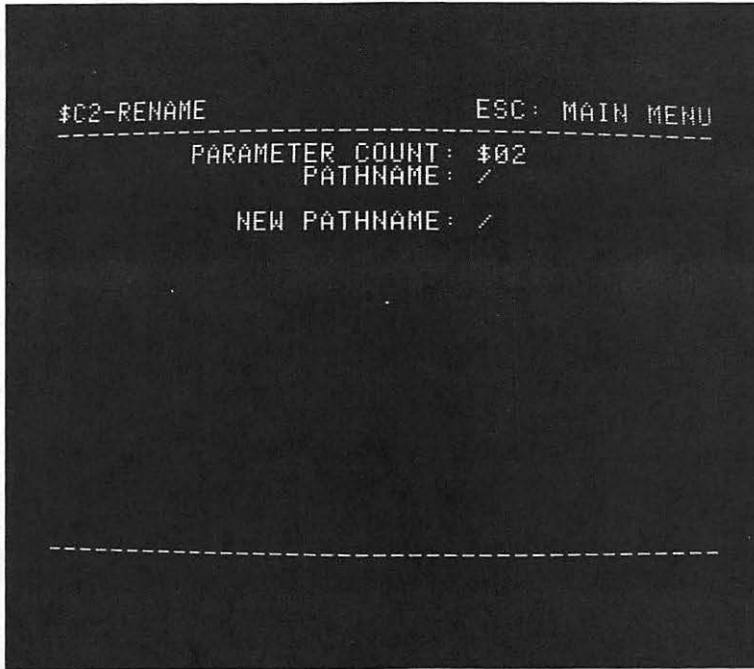


Figure 9.4. RENAME call.

**SET\_\_FILE\_\_INFO: (\$C3)**

This call sets the file's type, its modification date and time, and the way it may be accessed.

This call modifies information in a file's entry field. This call may be performed when a file is either open or closed. When a file is open, any change will not take effect until the file is closed and reopened.

The parameter count for this call is 7.

For the EXERCISER program you enter the pathname for the file you wish to destroy. For a call that you write, enter an address pointer to the ASCII string that contains the pathname. The string contains a count byte followed by the string itself. If the pathname begins with a slash, "/", character it is treated as a complete pathname of 64 characters or less. If not, it is treated as a partial pathname and the prefix is placed in front of the file name.

The ACCESS, FILE TYPE, and AUXILIARY TYPE are the same as with the CREATE call.

The 3-byte NULL FIELD is for the purpose of providing symmetry only with the GET\_\_FILE\_\_INFO call, discussed next.

The MOD DATE and MOD TIME are entered in the same form as shown for memory locations \$BF90 through \$BF93 as shown in Section 10.1.

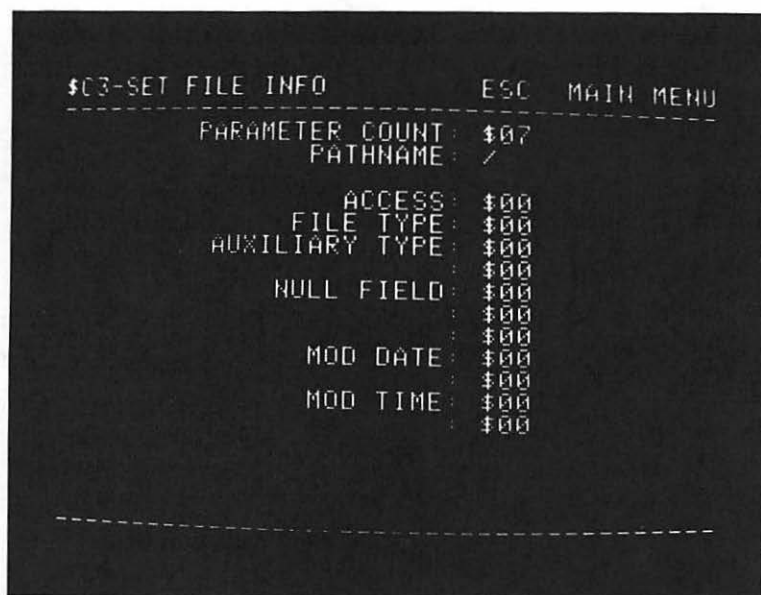


Figure 9.5. SET FILE INFO call.

**GET\_FILE\_INFO: (\$C4)**

This call is just the opposite of the previous one. This call will retrieve the file's type, its access method, the modification date and time, its size in blocks, and the way it is stored on a diskette.

The parameter count for this call is 10.

For the EXERCISER program you enter the pathname for the file you wish to destroy. For a call that you write, enter an address pointer to the ASCII string that contains the pathname. The string contains a count byte followed by the string itself. If the pathname begins with a slash, "/", character it is treated as a complete pathname of 64 characters or less. If not, it is treated as a partial pathname and the prefix is placed in front of the file name.

The ACCESS, FILE TYPE, and AUXILIARY TYPE are the same as with the CREATE call.

The MOD DATE and MOD TIME are entered in the same form as shown for memory locations \$BF90 through \$BF93 as shown in Section 10.1.

The CREATE DATE and CREATE TIME are entered in the same form as shown for memory locations \$BF90 through \$BF93 as shown in Section 10.1.

**ON\_LINE: (\$C5)**



```

$C4-GET FILE INFO                                ESC: MAIN MENU
-----
PARAMETER COUNT: $0A
  PATHNAME: █
    ACCESS: $00
    FILE TYPE: $00
  AUXILIARY TYPE: $00
    : $00
  STORAGE TYPE: $00
  BLOCKS USED: $00
    : $00
    MOD DATE: $00
    : $00
    MOD TIME: $00
    : $00
  CREATE DATE: $00
    : $00
  CREATE TIME: $00
    : $00
-----

```

Figure 9.6. GET FILE INFO call.

```

$C5-ON LINE                                ESC: MAIN MENU
-----
PARAMETER COUNT: $02
  UNIT NUMBER: $00 █
  DATA BUFFER: $00
    : $00
-----

```

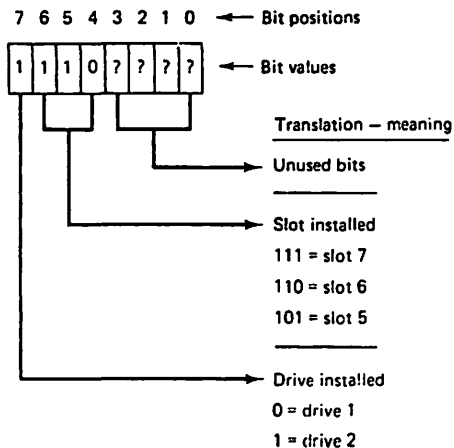
Figure 9.7. ON LINE call.

This call returns the slot number, drive number, and the volume names of all mounted volumes. This information is then placed in a user-defined and -supplied buffer.

The call is used to determine the names of all of the volumes that are currently active on your system or the name of a diskette in a particular slot and drive.

The parameter count for this call is 2.

The UNIT NUMBER byte specifies the slot of a disk drive device. The format of this byte is:

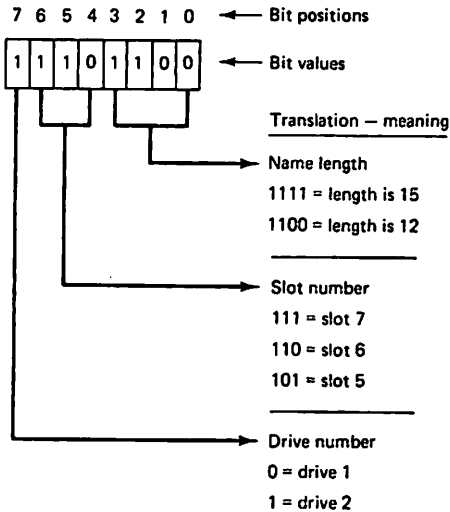


For example, the possible values for this byte are:

Slot		Drive 1		Drive 2
7	-	\$70	-	\$F0
6	-	\$60	-	\$E0
5	-	\$50	-	\$D0
4	-	\$40	-	\$C0
3	-	\$30	-	\$B0
2	-	\$20	-	\$A0
1	-	\$10	-	\$90

Note: These values may be easily verified from the above explanation.

The DATA BUFFER is a 2-byte pointer address, with low byte first, and gives the address of a buffer for returned data. This is organized into 16-byte records. If the UNIT NUMBER is zero then the buffer should be 256 bytes long. The first byte of a record is used to identify the length of the volume name and the device where it is stored. For example:



After this byte, the volume name is stored in the next 15 bytes.

The ON\_\_LINE call will return volume names that are not preceded by slashes. Therefore you must remember to place a slash character in front of the volume name before you can use it as a pathname.

#### SET\_\_PREFIX: (\$C6)

This call sets the pathname that is used by the operating system as the prefix. This prefix must include the volume directory name.

The parameter count for this call is 1.

For the EXERCISER program, you enter the pathname for the file you wish to destroy. For a call that you write, enter an address pointer to the ASCII string that contains the pathname. The string contains a count byte followed by the string itself. If the pathname begins with a slash, “/”, character it is treated as a complete pathname of 64 characters or less. If not, it is treated as a partial pathname and the prefix is placed in front of the file name.

#### GET\_\_PREFIX: (\$C7)

This call returns the value of the current system prefix.

This call will return the current system prefix.

The parameter count for this call is 1.

When writing your own call, you enter the address of the data buffer that is to contain the pathname. This buffer should be at least 128 bytes long.

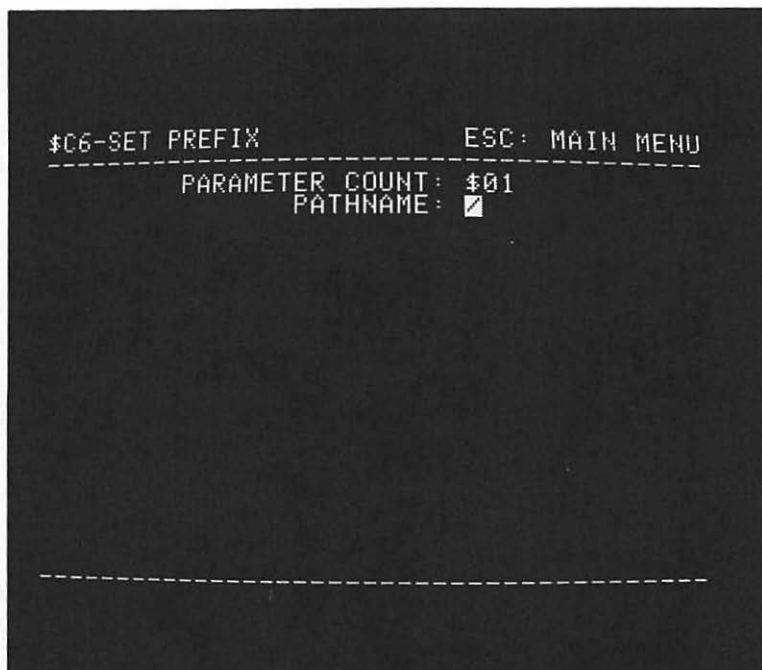


Figure 9.8. SET PREFIX call.

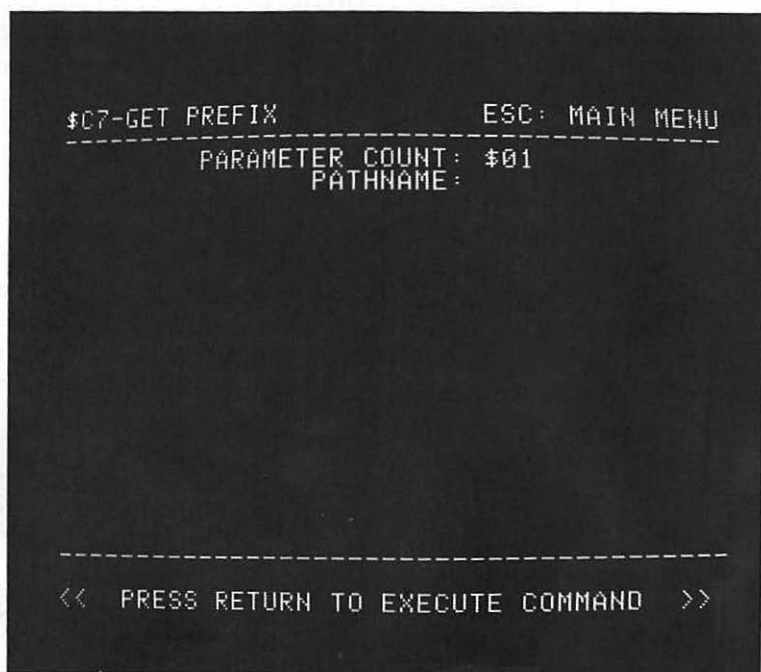


Figure 9.9. GET PREFIX call.

### 9.2.2. Filing calls

This section discusses calls that cause the transfer of data to or from files. In order to do this you must first OPEN communications with the operating system and the file.

#### OPEN: (\$C8)

This call prepares a file to be accessed. This call specifies the file name by its pathname that is to be opened. When this call is used, a reference number is returned by the call. The reference number is used with other calls to reference that file name. Further, there is a buffer allocated (IO-BUFFER) for the open file.

The parameter for this call is 3.

For the EXERCISER program you enter the pathname for the file you wish to destroy. For a call that you write, enter an address pointer to the ASCII string that contains the pathname. The string contains a count byte followed by the string itself. If the pathname begins with a slash character, “/”, it is treated as a complete pathname of 64 characters or less. If not, it is treated as a partial pathname and the prefix is placed in front of the file name.

The IO BUFFER entry is a 2-byte address pointer that indicates the start of the 1024-byte area in memory for the storage of all I/O that is to be done by the file indicated by

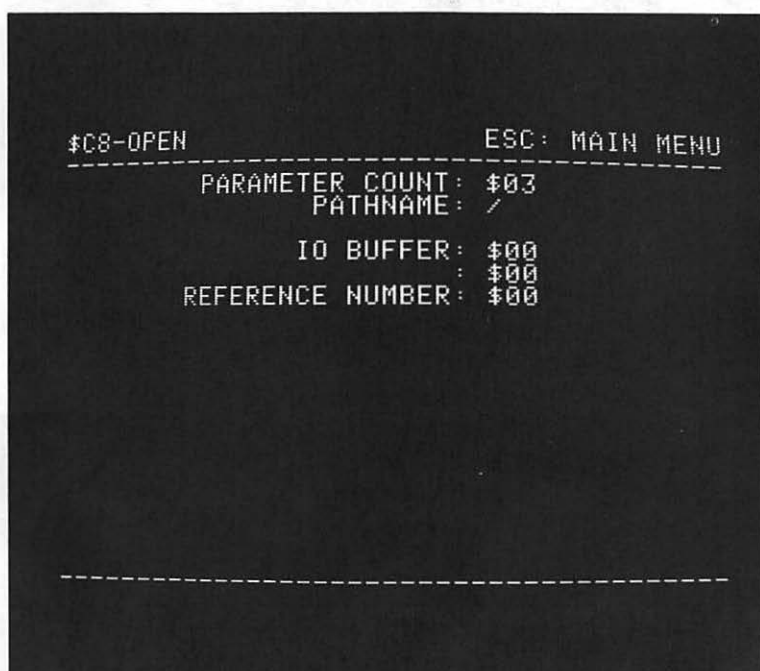


Figure 9.10. OPEN call.

the pathname. An I/O buffer address must start on a page boundary (i.e., multiples of \$100) that is not marked as in use by the system bit map.

The REFERENCE NUMBER is a 1-byte value assigned at the time of opening a file. You may use any number you choose, however, you must keep that number consistent when referencing the file that is open.

#### NEWLINE: (\$C9)

This call sets up the conditions necessary to read requests from users and to terminate those requests by typing a carriage RETURN.

This call allows you to either enable or disable the newline mode when working with an open file.

The parameter count for this call is 3.

The REFERENCE NUMBER is the one assigned during the open call to the MLI.

The ENABLE MASK is a 1-byte value that has the following meanings:

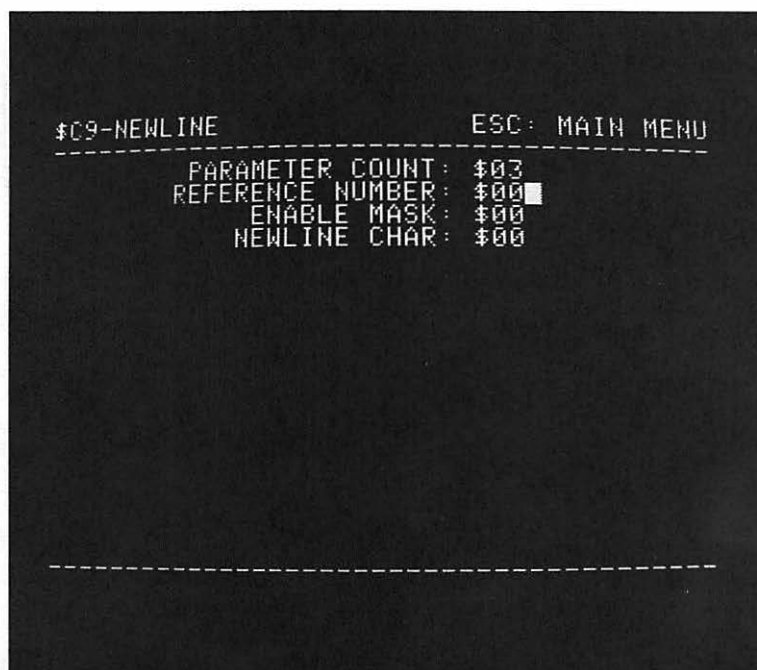


Figure 9.11. NEWLINE call.

<i>Byte</i>	<i>Meaning</i>
\$00	Disables the newline mode.
\$xx	Nonzero value enables newline mode.

The NEWLINE CHAR is a character that causes a read request to terminate when the NEWLINE mode is enabled and a match is found during the inputting of a character.

#### READ: (\$CA)

This call reads the specified number of characters from a file. Actually, this call does the actual transfer of the requested characters from the diskette to the memory buffer and then updates the current position of the mark in the file.

The parameter count for this call is 4.

The REFERENCE NUMBER is the one assigned during the open call to the MLI.

The DATA BUFFER is a 2-byte address pointer that points to the destination storage address for the data that is to be read from the opened file.

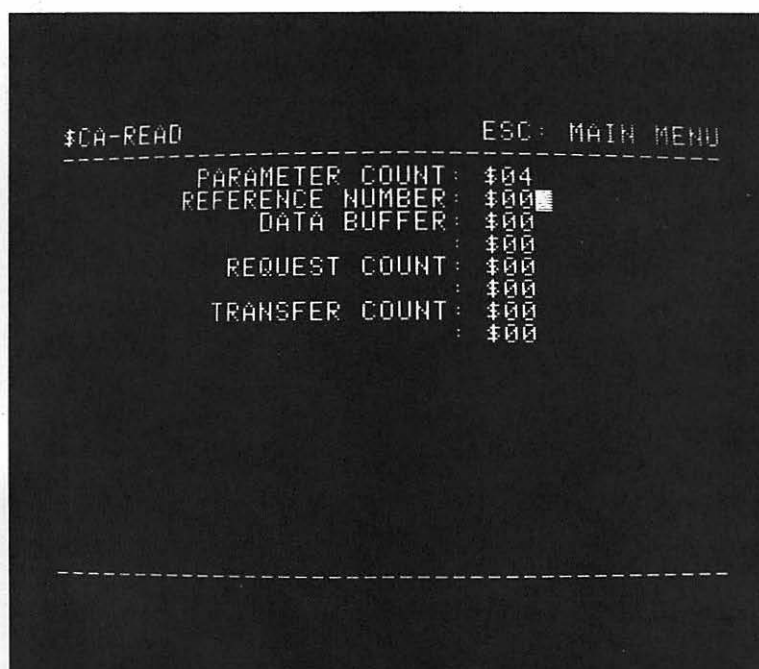


Figure 9.12. READ call.

The REQUEST COUNT is a 2-byte value that specifies the maximum number of bytes that are to be transferred from the file stored on a diskette to the area of memory pointed to by the DATA BUFFER address. This value is only limited by the amount of space from the DATA BUFFER address to the next page of memory shown in use by the system bit map.

The TRANSFER COUNT is a 2-byte value that tells you just how many bytes were actually transferred during the read operation. This value should equal the REQUEST COUNT or less if either the end-of-file marker was reached or if some other condition terminated the read operation.

#### WRITE: (\$CB)

This call writes the specified number of characters to the file. This call is the opposite of the READ call. This call will transfer characters from the file buffer to the diskette, update the current position mark in the file, and the end-of-file (EOF) marker, if it is required.

The parameter count for this call is 4.

The REFERENCE NUMBER is the one assigned during the open call to the MLI.

The DATA BUFFER is a 2-byte address pointer that points to the source storage address for the data that is to be written to the opened file.

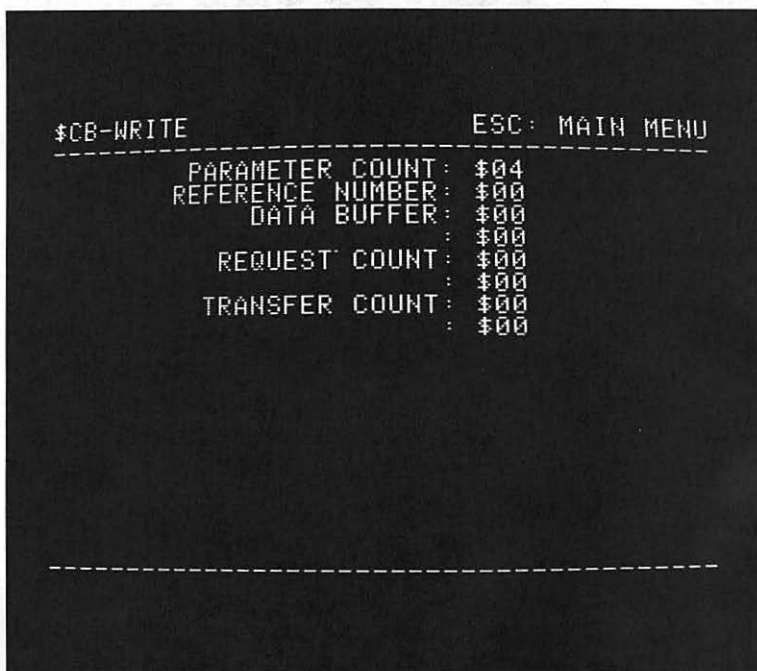


Figure 9.13. WRITE call.



The REQUEST COUNT is a 2-byte value that specifies the maximum number of bytes that are to be transferred to the file stored on a diskette from the area of memory pointed to by the DATA BUFFER address.

The TRANSFER COUNT is a 2-byte value that tells you just how many bytes were actually transferred during the write operation. This value should equal the REQUEST COUNT if no error occurs.

#### CLOSE: (\$CC)

This call closes the specified file. Any unwritten data from the file's buffer (IO-BUFFER) is transferred, the file's memory buffer (IO-BUFFER) is released, and the file's directory is updated, if that is necessary.

The parameter count for this call is 1.

The REFERENCE NUMBER is the one assigned during the open call to the MLI. If the REFERENCE NUMBER is zero, all open files at or above the current level number are closed.

#### FLUSH: (\$CD)

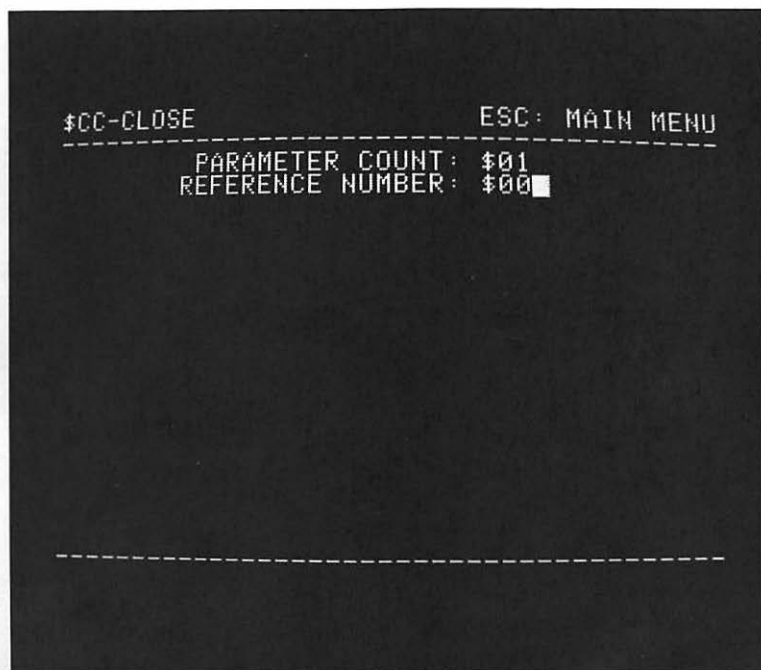


Figure 9.14. CLOSE call.

This call transfers any unwritten data from the file buffer to the file on a diskette of the open file.

The parameter count for this call is 1.

The REFERENCE number is the one assigned during the open call to the MLI. If the REFERENCE NUMBER is set to zero, then all open files at or above the current level will be flushed.

#### SET\_\_MARK: (\$CE)

This call changes the current position in the file. Current position means the place or position in the file where the next character will be either read from the file or written to the file.

The parameter count for this call is 2.

The REFERENCE NUMBER is the one assigned during the open call to the MLI.

The POSITION entry is a 3-byte value that specifies the absolute position in the file where the next read or write is to take place. The POSITION marker may not exceed the end-of-file marker.

#### GET\_\_MARK: (\$CF)

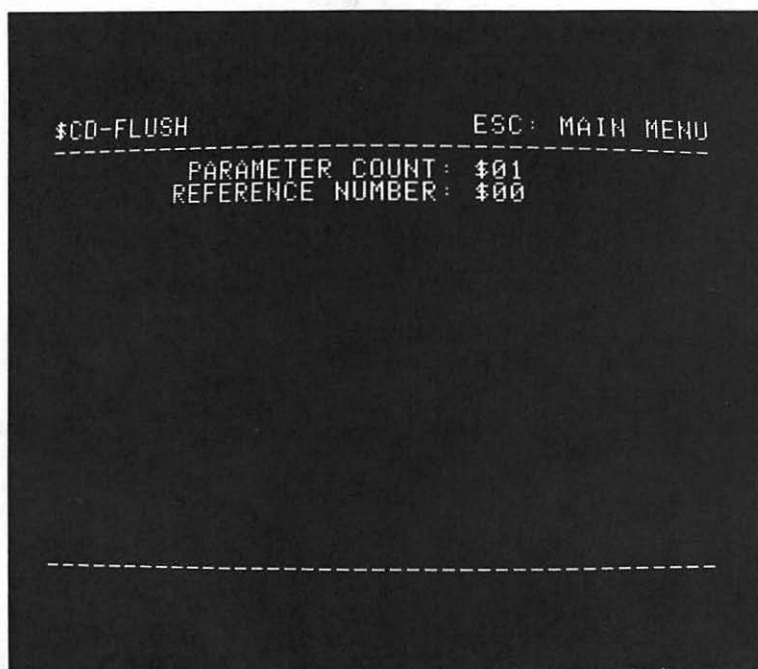


Figure 9.15. FLUSH call.

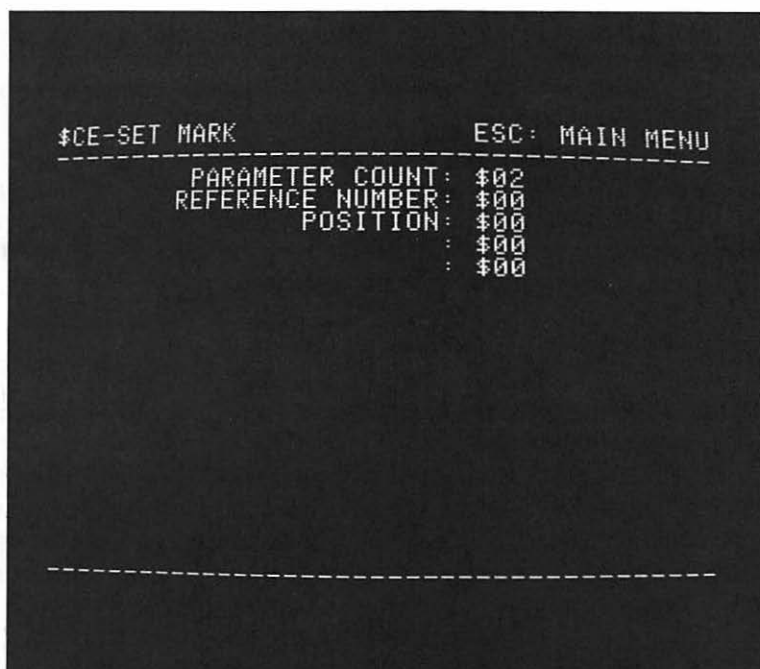


Figure 9.16. SET MARK call.

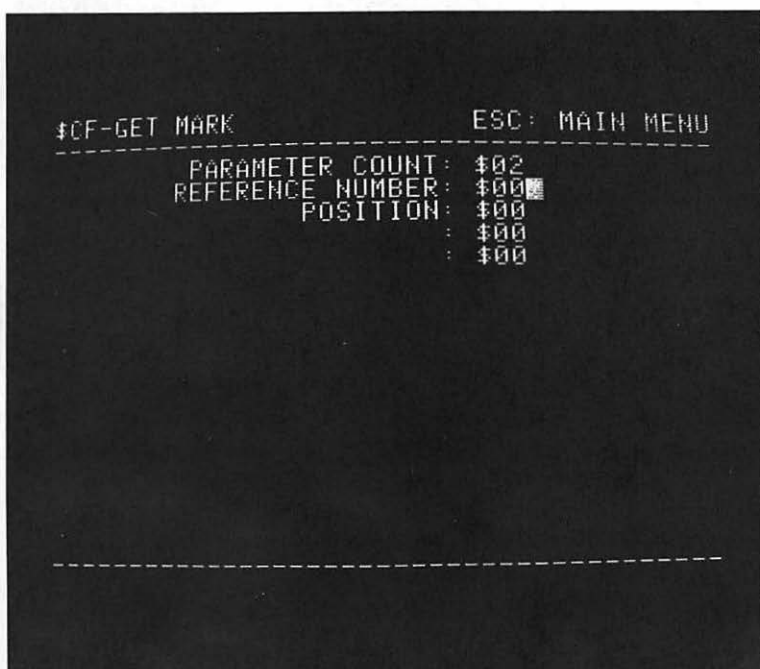


Figure 9.17. GET MARK call.

This call returns the current position in the file. This call tells you where you are in the file.

The parameter count for this call is 2.

The REFERENCE NUMBER is the one assigned during the open call to the MLI.

The POSITION entry is a 3-byte value that specifies the absolute position in the file where the next read or write is to take place unless it is changed by a SET\_\_MARK call. The POSITION marker may not exceed the end-of-file marker.

#### SET\_\_EOF: (\$D0)

This call will cause a change in the logical size of a file. The end of the file is changed.

The parameter count for this call is 2.

The REFERENCE NUMBER is the one assigned during the open call to the MLI.

The END OF FILE entry is a 3-byte entry that specifies where the logical end of the file is to be placed. It may be either greater than or less than the current end-of-file position. If it is greater, then the additional pages are marked as in use. If it is less, then the extra pages are released to the system.

#### GET\_\_EOF: (\$D1)

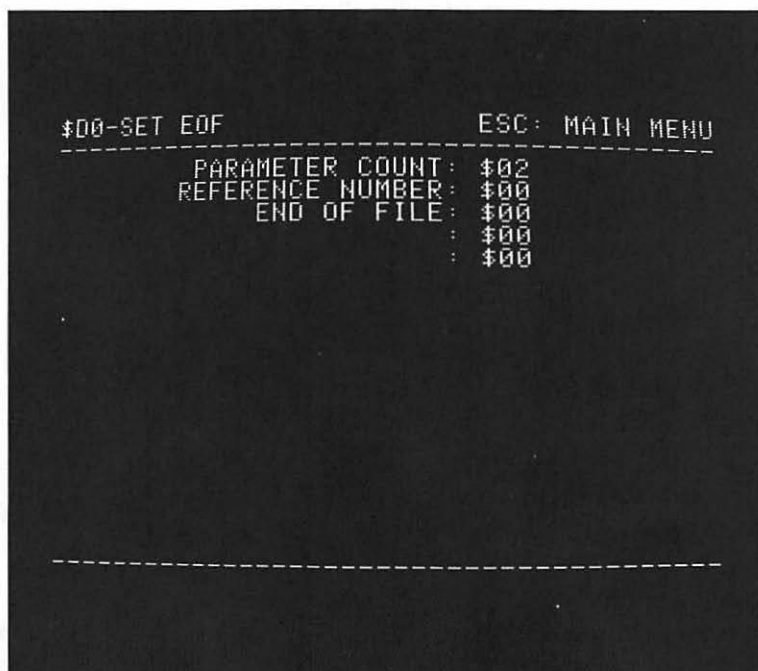


Figure 9.18. SET EOF call.

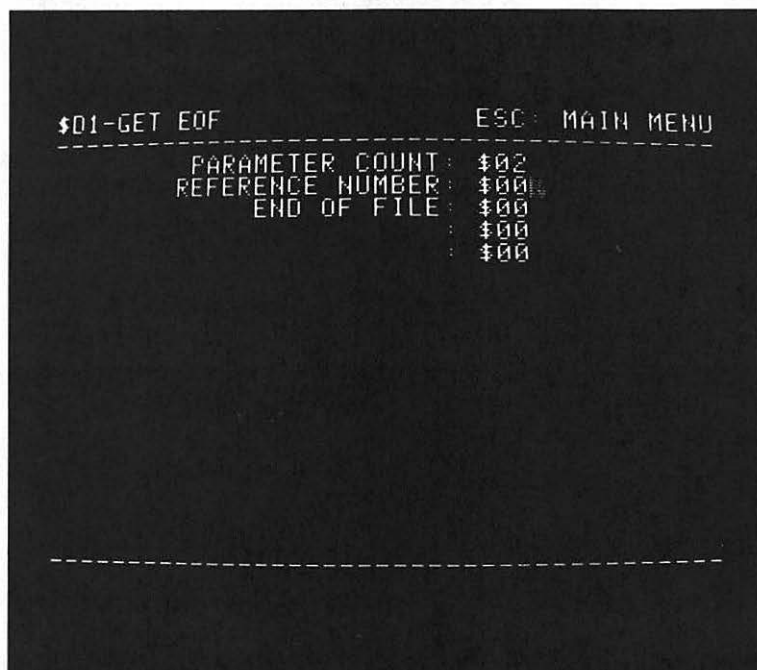


Figure 9.19. GET EOF call.

This call will tell you the logical size of the file.

The parameter count for this call is 2.

The REFERENCE NUMBER is the one assigned during the open call to the MLI.

The END OF FILE entry is a 3-byte entry that specifies where the logical end of the file is located. This value tells you the number of bytes that might be read from the open file.

#### SET\_\_BUF: (\$D2)

This call assigns a new place in memory for the file buffer.

The parameter count for this call is 2.

The REFERENCE NUMBER is the one assigned during the open call to the MLI.

The IO BUFFER is a 2-byte address pointer to the 1024-byte buffer. This buffer must start at a page boundary (multiples of \$100) and must not be marked as in use.

#### GET\_\_BUF: (\$D3)

This call will tell you the current location of the file memory buffer for an open file.

The parameter count for this call is 2.

The REFERENCE NUMBER is the one assigned during the open call to the MLI.

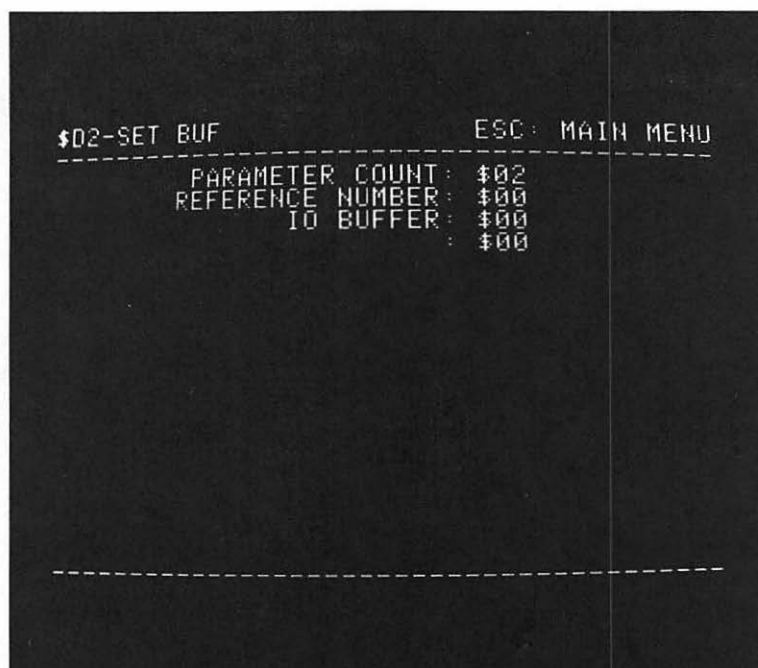


Figure 9.20. SET BUF call.

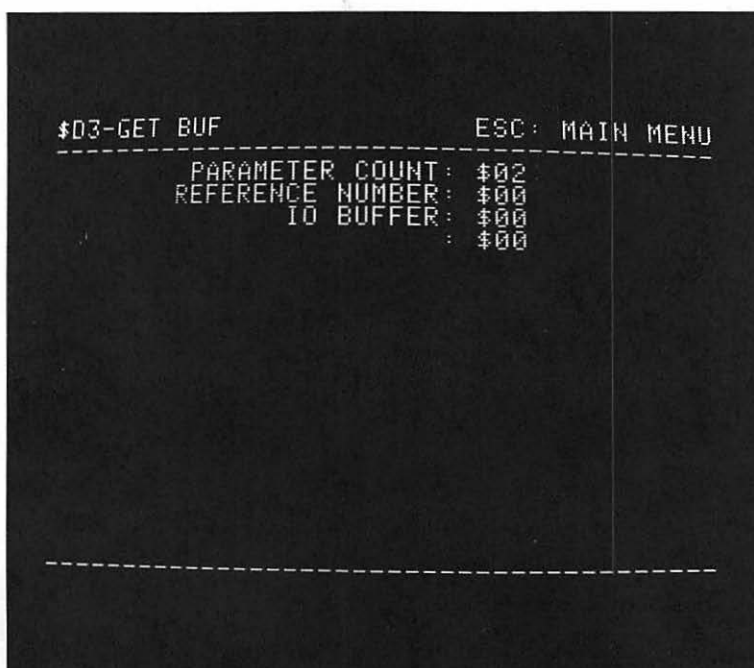


Figure 9.21. GET BUF call.

The IO BUFFER is a 2-byte address pointer to the 1024-byte buffer. This buffer must start at a page boundary (multiples of \$100) and must not be marked as in use.

### 9.2.3. System calls

This section discusses the calls that have to do with the system as a whole.

#### GET\_\_TIME: (\$82)

This call places the date and time data in the system date and time locations (\$BF90 through \$BF93). Of course you must have a clock/calendar card installed in your system and have the proper routine installed. If you have a Thunderclock installed in the recommended slot 4 then this call will work without you having to do anything but call it.

This call has no parameter list and is not in the EXERCISER program.

Since this call has no parameter list, it cannot generate an error. This call returns to memory location \$BF90 through \$BF93 the date and time as generated by the clock/calendar card, if one is installed in your system. The format of this information was shown earlier in the chapter.

When you power up your system with ProDOS, the clock/calendar card is looked for and the routine is installed. The recommended slot for the clock/calendar card is slot 4.

#### ALLOC\_\_INTERRUPT: (\$40)

This call places a pointer (address) to the interrupt-handling routine into the system interrupt vector table.

This call is not in the EXERCISER program.

The parameter count for this call is 2.

The INTERRUPT NUMBER may be any value from 1 through 4 as you assign. This number, like the REFERENCE NUMBER, is constant during the time when the interrupt is active.

The INTERRUPT CODE entry is a 2-byte address pointer to the first byte of the routine called when the system is polling in response to the activation of the interrupt.

#### DEALLOC\_\_INTERRUPT: (\$41)

\$82-GET TIME

ESC: MAIN MENU

---

Figure 9.22. GET TIME call.

**\$40-ALLOC INTERRUPT****ESC: MAIN MENU**


---

```

PARAMETER COUNT: $02
INTERRUPT NUMBER: $00
  INTERRUPT CODE: $00
                : $00

```

---

Figure 9.23. ALLOC INTERRUPT call.

This call removes the pointer (address) to the interrupt-handling routine from the system interrupt vector table.

This call is not in the EXERCISER program.

The parameter count for this call is 1.

The INTERRUPT NUMBER may be any value from 1 through 4 as you assign. This number, like the REFERENCE NUMBER, is constant during the time when the interrupt is active. This is the same number as when the interrupt was activated.

**9.2.4. Direct access calls**

With the MLI, you have the ability to read and write blocks of data on a diskette directly. These are for the purpose of performing diagnostics, repairs, and utilities directly on a diskette. These calls are not for the purpose of working with track and sectors as in DOS 3.3.

**READ\_BLOCK: (\$80)**

This call will read a specific block of 512 bytes of information from a diskette into a specified data buffer. This is different from the READ call.

**\$41-DEALLOC INTERRUPT****ESC: MAIN MENU**


---

```

PARAMETER COUNT: $01
INTERRUPT NUMBER: $00

```

---

Figure 9.24. DEALLOC INTERRUPT call.



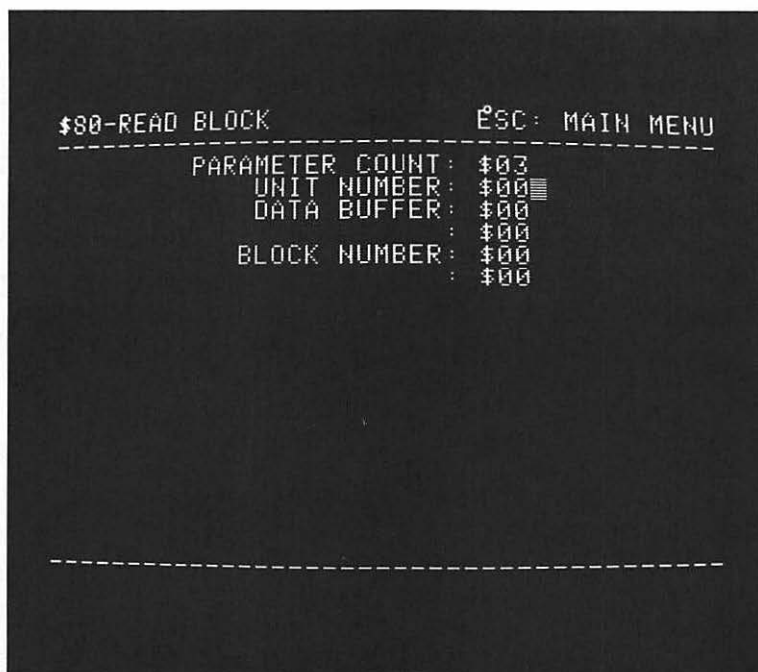


Figure 9.25. READ BLOCK call.

The parameter count for this call is 3.

The UNIT NUMBER byte specifies the slot of a disk drive device. The format of this byte was shown in the ON LINE call.

The DATA BUFFER is a 2-byte address pointer that points to the destination storage address for the data that is to be read from the diskette. The buffer must be at least 512 bytes long.

The BLOCK NUMBER is a 2-byte value that specifies the logical address on the diskette that is to be read. As it turns out, there is no general direct connection between tracks and sectors on a diskette and the logical block you are going to read. The translation between the two is done by the device driver for the device that is going to be read.

#### WRITE\_BLOCK: (\$81)

This call will write a specific block of 512 bytes of information to a diskette from a specified data buffer. This is different from the WRITE call.

The parameter count for this call is 3.

The UNIT NUMBER byte specifies the slot of a disk drive device. The format of this byte was shown in the ON LINE call.

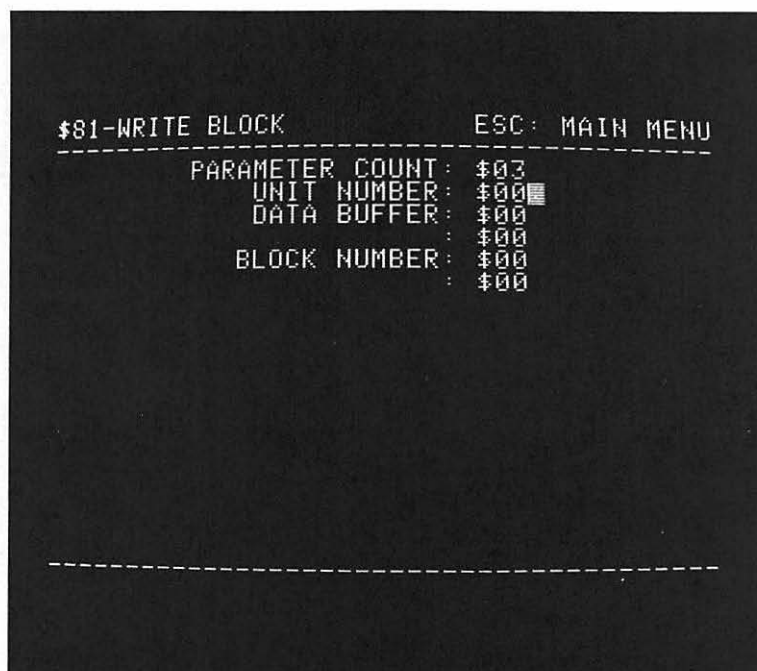


Figure 9.26. WRITE BLOCK call.

The DATA BUFFER is a 2-byte address pointer that points to the destination storage address for the data that is to be transferred to the diskette. The buffer must be at least 512 bytes long.

The BLOCK NUMBER is a 2-byte value that specifies the logical address on the diskette that is to be read. As it turns out, there is no general direct connection between tracks and sectors on a diskette and the logical block you are going to write. The translation between the two is done by the device driver for the device that is going to be doing the writing.

### 9.3. MLI ERROR CODES

The next tabular information lists all of the MLI error codes that could be returned from MLI calls.

##	Hex	Description
0	\$00	No error present.
1	\$01	A non-existent command was issued. Bad system call number.

4	\$04	Bad system call parameter count. This error occurs only if the parameter list is not constructed properly.
37	\$25	Interrupt vector table full. Only four are allowed to be active for interrupt processing simultaneously.
39	\$27	I/O error. General error number.
40	\$28	No device detected or connected.
43	\$2B	Diskette is write-protected.
46	\$2E	Disk switched while file on previous diskette was OPEN.
64	\$40	Invalid pathname syntax.
66	\$42	File Control Block table full. Only eight files may be open at one time.
67	\$43	Invalid reference number. The value parameter does not match the reference number of any open file.
68	\$44	Path not found.
69	\$45	Volume directory not found.
70	\$46	File not found.
71	\$47	Duplicate filename encountered.
72	\$48	Overrun error.
73	\$49	Volume directory full.
74	\$4A	Incompatible file format.
75	\$4B	Unsupported storage__type.
76	\$4C	End of file has been encountered.
77	\$4D	Position out of range.
78	\$4E	Access error.
80	\$50	File is open.
81	\$51	Directory count error.
82	\$52	Not a ProDOS disk.
83	\$53	Invalid parameter.
85	\$55	Volume Control Block table error.
86	\$56	Bad buffer address.
87	\$57	Duplicate volume.
90	\$5A	Bit map disk address impossible.

## 9.4. WRITING A SYSTEM PROGRAM

This section will discuss how to go about writing a system program. It is not meant to be exhaustive, but is meant to give you a reasonably good starting place.

First of all, a system program is any program that uses the ProDOS MLI and follows the standard system program rules. Each program must:

—contain code to move the program from its loaded position in memory to its final location for execution. This, of course, may not be necessary.

- contain code that places the version number in the system global page.
- contain code to perform a quit and possibly switch to another program.

A system program always loads into memory at location \$2000. When your system program is first started, this program must be the first file of the startup diskette with the name xxx.SYSTEM. This program will have a file type of \$FF (SYS).

It is recommended that you look carefully over the system global page, \$BF, locations \$BF00 through \$BFFF, for other information and values that you may use to advantage.

When you have finished executing the current system program, you will want to provide code to execute possibly another system program. It is recommended that you use the following method for switching:

1. Close all open files.
2. Prompt the operator for the name of the next system program or hard code the new name.
3. Open the new file containing the system file.
4. Find the length of the new file using the GET EOF call.
5. Release the memory used by your current system program.
6. Load into memory the entire file length, starting at location \$2000.
7. Close the system program file.
8. Store the pathname or partial pathname starting at memory location \$280. Remember to place the count byte in front of the name.
9. Execute a JMP (jump) to location \$2000.

## SUMMARY

This chapter showed you some of the inner workings of ProDOS. A number of useful memory locations were shown, along with their translations. This was done to give you a better feel for this new operating system and how things are done inside the code.

The next major section introduced you to the Machine Language Interface (MLI). As you can see from the numerous calls available, you have almost complete control over your system through the use of this mechanism. You should be able to write calls in assembly language to go with your assembly language applications.

Through the use of the EXERCISER program, you should be able to gain competence with the powerful MLI capability.

The last section of this chapter discussed writing a ProDOS system program.

## QUESTIONS

1. Describe the differences between the command tables for DOS 3.3 and ProDOS.
2. How can the EXERCISER program help you?
3. How are the date and time stored in memory?

4. Describe how the system bit map works. Why is it important that you understand this?
5. Describe how the CREATE call works.
6. Describe the major parts of the MLI.
7. What code is required to make an MLI call?
8. Describe how to use the machine identification byte in an Applesoft II BASIC program.
9. Describe the GET\_\_FILE\_\_INFO call.

# 10. BASIC PROGRAMMING SYSTEM

*Top-down segmented programming  
is one way to make programming  
the enjoyable activity it  
should be.*

*McGowan and Kelly, 1975*

## 10.0. OVERVIEW

This chapter will discuss the creation of a small management information system designed primarily to handle only one type of information. In this case, your Christmas card mailing list.

Section 10.1 discusses the requirements to be met in designing any program and this program in particular.

Section 10.2 outlines the major program modules to be included either as in-line main driver code or as subroutine modules.

Section 10.3 gives you the completed program listing along with a line-by-line description of the code.

A number of programming principles and techniques will be discussed without giving you all of the motivations for using them. These were covered in detail in my previous books, as mentioned in Chapter 1.

The primary purpose of this chapter is to show you how to use ProDOS within programs, dynamically configure screen presentations to your system, intelligently determine

your system configuration, and create a well-structured program as well as to provide you with a reasonably pragmatic program.

Since the program given in this chapter seems very long and complex, each of the major sections will be discussed individually. You will be asked to enter the code discussed in that section. In this way, entering the entire program becomes rather painless. The program is complete; however, there are a number of places that additional code could be written.

This is especially true of the disk error trapping. The error trapping codes and messages have been placed in the code, but not all possible errors have been accounted for. The rest of the job is left up to you.

Furthermore, there are a couple of areas where the code could be improved. That task is also left up to you. It is good practice.

## 10.1. DESIGNING THE PROGRAM

This section discusses the process by which a digital computer solves, or rather, aids in the solution of a problem. For the programmer, this involves the following:

1. Select problems that are appropriate to the ability and computing power of the computer system.
2. Describe the problem requirements precisely.
3. Design a solution to the problem.
4. Describe the problem solution in a language intelligible to the computer system.
5. Confirm the correctness of the program solution.
6. Document the program completely.

The appropriate problem selection for solution on a microcomputer is as important and as difficult as the problem description and analysis because this involves knowledge of the particular computer system, its abilities, peculiarities, and capabilities. However, two facts are immediately apparent and helpful.

First, the problem is of sufficient magnitude that manual solution is time-consuming and is one that recurs often. If the problem is simple and nonrepetitive, then a computer solution may cost more than it is worth. The problems given in this text for solution and instruction are, for the most part, reasonably simple, so that you may develop competence in programming, and acquire confidence in your ability to program.

Second, the computer can only assist in the solution of problems for which a precise and detailed definition exists. If the designer of a problem does not define precisely the problem and instructional details to the computer, there is no chance that a precise and reliable solution will be realized.

The classic example of this is the game of chess. The computer is asked to "play chess" against a human opponent. Over the years, these computer programs have become better and more comprehensive as the abilities of the programmers have become better, the problem definition has become more precise, computer capability to symbolically manipulate the "data" has increased, and the computer software "intelligence" level has matured.

To describe the problem requirements precisely, all inputs, all outputs, files, file structures, interactions of files, correlation of data, intermediate and final calculations, all reports, screen formats, error conditions, and trapping **MUST** be defined in detail. This activity is paramount, because if it is not done correctly, the remainder of the problem solution implementation will not be correct.

From the analysis done in the above paragraph, design of the problem solution may be accomplished precisely, correctly, and in detail. The steps necessary for solution, their order of accomplishment, and the expected results may now be described in terms necessary for problem solution in the form of a computer program, and implemented using any one of a number of languages recognized by the chosen computer. You will use Apple's Applesoft II BASIC language for problem solution since that is the language supported by ProDOS.

After coding the program, it is necessary to confirm the correctness of the solution. This entails more than just debugging the syntactic, semantic, and logic errors. It also entails checking ranges of inputs, correctness of responses, and routines, subroutines, and coding modules. Checking correctness is time-consuming, tedious, and exhausting.

Finally, you are ready to write down the problem documentation and your implementation of the solution. This usually requires two documents: the run book and the program documentation. At this point, you put all the descriptions, analyses, program hard copy, example reports, and a detailed narrative of how to run the program together, in an organized manner. Other individuals may use your problem solution with confidence now.

Assuming that you do all the above formally, you will find that the actual amount of time spent coding the problem is less and the task is much less frustrating.

At this point, let's define the problem to be accomplished.

**A Problem.** You are to write a computer program that will provide you with a Christmas card mailing list saved on a diskette. It should keep track of the names and addresses for the cards you mail and the years you receive cards from the members of the list.

#### **B Requirements.**

Allow for differing system configurations

Create a Christmas card mailing list

Delete outdated card files

Save to a diskette the individual;

    Name

    Address

    Telephone number

    Past card receptions

Print mailing labels on printer

Print list contents on printer

Add new records

Delete old records

Review list records on video screen

Change a record contents



End the program  
 Check for correctness  
 Document the solution

- C Describe solution.* The program written will meet all of the requirements by being able to operate upon one record at a time and by allowing the operator to manipulate each record completely. Top-down structured programming techniques will be used, along with implementation of routines required multiple times as subroutines.
- D Program and language.* The Christmas card list program will use the ProDOS operating system and the Applesoft II BASIC language. The listing of this program is shown in Section 10.3.
- E Confirm correctness.* Trying to determine that a program, especially a large one, is correct for all data is very difficult and time-consuming, and is an often neglected area.

This phase of the computing process seems to be neglected by almost everyone—in their writings, in program development, and in documentation. In all but the most simple programs, many errors will be made in each phase of the program's development. Anyone who intends to use a computer will just have to accept this as a fact.

Typically, more than half of the total time, effort, and expense of any program development is spent tracking down errors, flaws, or mistakes. This activity is done during the testing (debugging) phase. Even with all the effort that is expended in the testing phase, undiscovered flaws will still crop up, perhaps years later. This is so serious that large portions of our society today have very little confidence in computers and the computing process.

Articles appear daily in newspapers, on television, and in magazines extolling the apparent inadequacy of computers. However, in almost all cases, the fault lies not with the computer, but with a program that was improperly designed or inadequately tested.

There are a number of levels of correctness that the developer of a program may observe. These are shown below:

1. There are no syntax errors in the code.
2. There are no semantic errors in the code.
3. The program works correctly for the test data.
4. The program works correctly for all valid test data.
5. The program works correctly for all possible sets of data that meet problem specifications.
6. The program works for all possible data, rejecting invalid data, and giving accurate answers.

Almost all programs can gain the fourth level of correctness. Today we are seeing programs that have reached the fifth level of correctness. This was not true just a couple of years ago. All programs on the market should operate with top-level correctness. However, especially with microcomputers, that is extremely difficult to achieve because of:

- the physical limitations of the microcomputer, in memory size and language inadequacy.
- the reluctance of programmers to spend the time, effort, and money to account for error-free code.

It seems obvious that the first limitation really is not a valid argument, because of the increased memory sizes available today and the proliferation of languages for microcomputers. The second reason is really the crux of the problem.

Once programmers and manufacturers spend the time, effort, and money, top-level programs will, in fact, be seen in the marketplace.

This next paragraph promulgates a set of principles that, if followed, should help all writers of programs to produce programs that are aesthetically satisfying and help produce correct results.

*“Now hear this . . .”*

Do NOT panic at any time.  
 Define the problem completely.  
 Proceed in a top-down manner.  
 Structure all code.  
 Write correct syntax.  
 Start the documentation early.  
 Forsake all other approaches.  
 Code in logical units.  
 Use GOTOs sparingly.  
 Use GOSUBs liberally.  
 Make all output pleasant.  
 Do NOT violate any of these principles.

*F Document solution.* It is always a good idea to document your programs, because two years from now you probably will not remember what you wrote or how to operate that program. You should explain minimally how to operate each major module of any program.

For the program you are about to enter and run, there are a few very simple conventions that are to be followed when you operate the finished program. When you are answering a question or making a selection you will have a normal cursor presented on the video screen. In all of those cases it will only take a single keystroke as a response.

In those cases where you are presented with a blinking cursor that is either a plus sign, “+”, or an asterisk, “\*”, you will make your entry followed by a carriage return character, RETURN.

As you are operating this program, you will notice that the upper left hand corner of the screen states that the typing of an ESC key allows you to back out of the present evolution you are performing. The typing of an ESC key effectively nullifies the present operation. You are allowed to use the ESC key to back up or cancel the operation in most cases.

The only major exception to this general rule is when you are configuring your system (menu selection 1). When you are configuring your system, typing the RETURN key for each entry selects the default value as shown on the screen.

For each selection made from the main menu, you are asked to enter or verify the correctness of the prefix as shown on the screen. Then you are asked to enter the file name for the file with which you wish to work.

## 10.2. PROGRAM MODULES

There are, in any program, a number of evolutions that need to be performed at different times, under different conditions, within the program. Normally, these items are implemented as subroutines placed in the program outside of the main driver line of code. It then becomes a simple matter to call (GOSUB) the subroutine to accomplish what is required.

This section discusses a number of these routines and subroutines, what they accomplish, and how they add capability to your program.

### 10.2.1. The skeleton program

One of the primary concerns of programmers developing programs for the commercial market is to make their programs user-oriented. This means that the program should lead the user through the program with prompting. Further, the program should be easy for the user to operate.

In attempting to get the operator to make correct selections, a MENU should be provided from which the user will select the processing desired. A number of advantages are gained by using a menu:

- Convenient for the operator.
- User oriented.
- Easy to program.
- Operator training is easy.
- Error trapping is simple.
- Easy to implement in code.

Programs that have a menu from which to make selections should place the menu in the main driver logic as the controlling force in the program.

Since the menu takes on such importance, it should be well trapped so that any selection made is legal and valid. Further, make sure that the operator really wants to accomplish that particular evolution. In this way the programmer can be fairly well assured that his major selection process will provide for correct initial processing selection. If the program goes wrong here, the operator gets a very bad initial impression.

The SKELETON (XMAS. PROG) program in Figure 10.2 shows the general organization of a BASIC program that takes advantage of the characteristics of the Applesoft II BASIC interpreter as implemented on the Apple II Plus and Apple IIe, and the supported constructs. The major logic flow of the program is shown below:

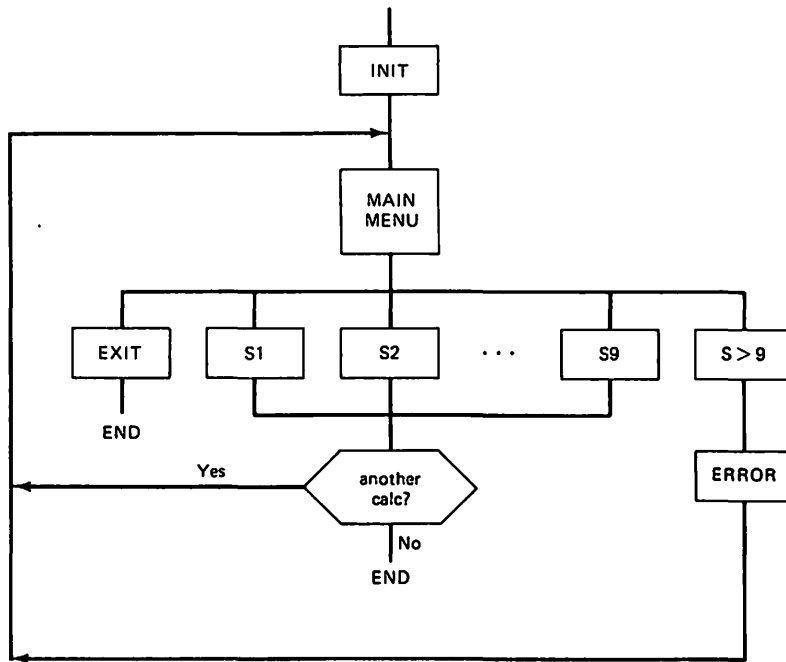


Figure 10.1. Skeleton logic flow.

Please refer to the XMAS.PROG program for the following discussion. Specific line numbers have been used for the placing of code. In this way your program takes on a clean organization.

<i>Lines</i>	<i>Description</i>
Lines 1-9	Identifies the program as to the author, date, program name, language, and computer.
Line 10	Sets up a clean machine.
Line 15	Free up all unused memory space.
Lines 20-490	DIMENSION statements, for defining variables, etc.
Line 495	Provides a hard branch to the main driver.
Lines 500-8995	Reserved for holding general subroutines that are called often during the execution of the program.
Lines 9000-9999	Holds the main driver code. This includes menus, menu trapping, and returns for further calculations, etc.
Lines 10000+	Holds the major subroutine blocks where programs will spend a large portion of time.
Lines 60000+	Holds all of the disk and language error trapping that is to be done.

This general arrangement for a program takes advantage of the best characteristics of modularity, speed, construction, etc. It further illustrates a program that is organized in a structured manner, using various constructs, and above all is easy to change, maintain, and "debug." It is felt that this type of organization has many more advantages than shortcomings.

Please realize that the line numbers assigned are NOT sacred. The point is that some thought concerning the physical arrangement of a program can and will pay big dividends in coding, running, and debugging the program.

You now should power up your system with the SCRATCH.DISK you made earlier. Type in this program segment and save it with the name XMAS.PROG:

]NEW

]LIST

```

1 REM ***** XMAS.PROG *****
2 REM *
3 REM * WRITTEN BY:JL CAMPBELL
4 REM *      DATED:MM/DD/YY
5 REM *
6 REM * LANGUAGE:APPLESOFT II
7 REM * COMPUTER:APPLE II
8 REM *
9 REM *****
10 TEXT : CLEAR : HOME : POKE 216,0:D$=CHR$(4)
15 PRINT D$;"FRE"
20 DIM M$(10) :REM * DIM STMTS & VAR. DEF.
80 FOR I = 1 TO 9: READ M$(I): NEXT I
82 DATA 1 -System set-up,2 -Create a file,3 -Delete a
   file
84 DATA 4 -Adds a record,5 -Edit a record,6 -Delete re
   cord
86 DATA 7 -Review record,8 -Prints report,9 -Prints lab
   els
440 REM *
495 GOTO 9000
497 REM *****
498 REM * GEN'L SUBR *
499 REM *****
500 INPUT "Which one? ==> ";: GET N$: PRINT N$:N = VAL (N$):NA = ASC (
   N$): RETURN
510 VTAB 22 : CALL - 958 : HTAB 13: INVERSE : PRINT
   "ILLEGAL ENTRY!"; CHR$ (7): NORMAL
520 FOR L = 1 TO 1000: NEXT L : RETURN
530 VTAB V: HTAB H: PRINT "Date: "; LEFT$ (T$,5): RETURN
540 VTAB V: PRINT "*CURRENT PREFIX = ";PR$: RETURN
550 PRINT "Is that correct? (Y/N)";: GET Q$: PRINT Q$: RETURN
560 PRINT "Press any key to continue...";: GET Q$: PRINT :
   RETURN
570 VTAB V: PRINT "*ESC=ESCAPE": RETURN
580 VTAB V: PRINT "DO YOU WANT THAT? (Y/N) ": RETURN

```

```

590 PRINT "Any changes? (Y/N) ";: GET Q$: PRINT Q$: RETURN
600 PRINT "Another entry? (Y/N) ";: GET Q$: PRINT Q$:
    RETURN
8950 FOR I = 1 TO 39: PRINT "*";: NEXT I: PRINT : RETURN
8960 PRINT "*";: HTAB 39: PRINT "*": RETURN
8999 REM * MAIN DRIVER
9000 HOME : PRINT : PRINT TAB( 11)"*** XMAS LIST ***":
    PRINT
9010 VTAB 4: PRINT "Main menu:";: HTAB 29: PRINT "Date: ";
    LEFT$(T$,5): PRINT "---- ----";: HTAB 29: PRINT "--
    --- ----"
9015 FOR I = 1 TO 9: VTAB 5 + I: PRINT TAB( 11)M$(I): NEXT
    I
9020 VTAB 16: PRINT TAB( 11)"0 -Exit program"
9025 IF TF = 1 THEN VTAB 6: PRINT "DO THIS"
9030 VTAB 18: CALL - 958: GOSUB 500
9040 IF N > 0 AND N < 10 THEN 9100
9050 IF NA = 48 THEN 9990
9060 GOSUB 510: GOTO 9030
9100 VTAB 5 + N: HTAB 5: PRINT " ==> ";: PRINT M$(N);" <
    == "
9110 VTAB 20: CALL - 958: GOSUB 550
9120 IF Q$ = "Y" OR Q$ = "y" THEN 9200
9130 IF Q$ = "N" OR Q$ = "n" THEN 9150
9140 GOSUB 510: GOTO 9110
9150 VTAB 5 + N: PRINT " ";M$(N);" "
9160 GOTO 9030
9200 ON N GOSUB 10000,15000,20000,25000,30000,35000,40000,
    45000,50000
9210 GOTO 9000
9990 HOME : VTAB 10: PRINT "GOODBYE FOR NOW..."
9995 END
10000 HOME : GOSUB 8950: GOSUB 8960
10010 PRINT "*";: HTAB 13: PRINT "SYSTEM SET-UP";: HTAB 39
    : PRINT "*"
14995 SW = 0:RETURN
15000 HOME : GOSUB 8950: GOSUB 8960
15010 PRINT "*";: HTAB 13: PRINT "CREATE A FILE";: HTAB 39
    : PRINT "*"
19995 SW = 0:RETURN
20000 HOME : GOSUB 8950: GOSUB 8960
20010 PRINT "*";: HTAB 13: PRINT "DELETE A FILE";: HTAB 39
    : PRINT "*"
24995 SW = 0:RETURN
25000 HOME : GOSUB 8950: GOSUB 8960
25010 PRINT "*";: HTAB 13: PRINT "ADDS A RECORD";: HTAB 39
    : PRINT "*"
29995 SW = 0:RETURN
30000 HOME : GOSUB 8950: GOSUB 8960
30010 PRINT "*";: HTAB 13: PRINT "EDIT A RECORD";: HTAB 39
    : PRINT "*"
34995 SW = 0:RETURN
35000 HOME : GOSUB 8950: GOSUB 8960

```

```

35010 PRINT "*";: HTAB 13: PRINT "DELETE RECORD";: HTAB 39
      : PRINT "*"
39995 SW = 0:RETURN
40000 HOME : GOSUB 8950: GOSUB 8960
40010 PRINT "*";: HTAB 13: PRINT "REVIEW RECORD";: HTAB 39
      : PRINT "*"
44995 SW = 0:RETURN
45000 HOME : GOSUB 8950: GOSUB 8960
45010 PRINT "*";: HTAB 13: PRINT "PRINTS REPORT";: HTAB 39
      : PRINT "*"
49995 SW = 0:RETURN
50000 HOME : GOSUB 8950: GOSUB 8960
50010 PRINT "*";: HTAB 13: PRINT "PRINTS LABELS";: HTAB 39
      : PRINT "*"
54995 SW = 0:RETURN

```

Figure 10.2. Skeleton program.

The following is a line-by-line description of the code shown in Figure 10.2.

<i>Line</i>	<i>Description</i>
Line 1-9	Identifies the program.
Line 10	Sets up a clean machine. TEXT = Places your computer into the text mode. The cursor is placed at the bottom of the screen. CLEAR = Clears all variables and the processor stack. HOME = Places the cursor at the upper left corner of the screen. POKE 216,0 = Turns off the ONERR flag. D\$ = CHR\$(4) = Set up the disk command identifier character.
Line 15	Performs a fast garbage collection on memory.
Line 20	Dimension statement that sets aside memory storage locations for variables.
Line 80	Menu READ statement that reads and stores the main menu.
Lines 82-86	Main menu data that is to be read.
Line 495	Hard branch to the main driver code.
Line 500	General prompting subroutine for operator response.
Line 510	General error handler for operator entry errors.
Line 520	General delay loop.
Line 530	Routine to print the date. Notice that by defining the <i>V</i> and <i>H</i> values before entry into routine, you may place the date anywhere on the video screen.
Line 540	Print the current prefix on the video screen.
Line 550	General prompting to verify the correctness of an operator response.
Line 560	General instruction for the purpose of holding a screen and requesting operator to type any key.

Line 570	Print the ESC capability of the program.
Line 580	Request verification of the performance of an evolution.
Line 590	Request the verification of any changes to be performed.
Line 600	Request to operator for another entry.
Line 8950	Print a line of asterisks across screen.
Line 8960	Print asterisks at each edge of screen.
Line 9000	Clear the screen and print main menu title on screen.
Line 9010	Print "Main menu" and the date at top of the screen.
Line 9015	Print the main menu selections on the screen.
Line 9020	Print program exiting selection.
Line 9030	Locate and print the general prompting question.
Line 9040	Test operator's response for a valid selection value.
Line 9050	Test response for an exiting request.
	If true, exit the program.
Line 9060	Activate error message and then branch to line 9030.
Line 9100	Prompt operator to validate correctness of selection.
Line 9110	Locate cursor and ask operator to validate selection made.
Line 9120	Test for a Y answer.
	If true, branch to line 9200.
Line 9130	Test for an N answer.
	If true, branch to line 9150.
Line 9140	Activate error message and then branch to line 9110.
Line 9150	Reprint menu selection after clearing previous selection.
Line 9160	Make a hard branch to line 9030.
Line 9200	Depending upon the selection made, branch to the appropriate code section.
Line 9210	Make a hard branch to the start of the main driver code.
Line 9990	Clear the screen and print ending message on screen.
Line 9995	End the program.
Lines 10000-14995	Code section to set up your system.
Lines 15000-19995	Code section to create a new file on a diskette.
Lines 20000-24995	Code section to delete a file from a diskette.
Lines 25000-29995	Code section to add a new record to the file.
Lines 30000-34995	Code section to edit a record in a file.
Lines 35000-39995	Code section to delete a record in the file.
Lines 40000-44995	Code section to review individual records in the file.
Lines 45000-49995	Code section to print list contents.
Lines 50000-54995	Code section to print labels.

The code you have seen is the main driver code and a number of single line subroutines that support the main driver and other code in the XMAS.PROG shown completely in Section 10.3.



### 10.2.2. Number and string module

The number filter is probably the more important of the next two routines because entering numbers into any program is done much more often than the entry of string data. Therefore, it becomes necessary and important that numeric data be accurate, correct in form, and not contain alphabetic characters. If the person entering data is used to typing on a standard typewriter, one of the more prevalent errors committed is to type an "O" instead of a numeric zero and to type an "l" in place of a numeric one. These mistakes can cause horrible results when trying to do mathematics on data. Type in the Number Filter routine and save it on a diskette as NUMBER.FILTER and then place it into your XMAS.PROG.

```

1495  REM * NUMBER FILTER
1500  IX = 1: B$ = "": SW = 0
1505  VTAB V: HTAB H + IX
1510  PRINT "+";
1515  VTAB V: HTAB H + IX
1520  GET X$
1525  IF ASC (X$) = 13 THEN 1590
1530  IF ASC (X$) = 8 THEN 1565
1535  IF IX > 1 THEN 1545
1540  IF ASC (X$) = 45 THEN 1555
1545  IF ASC (X$) = 46 AND SW = 0 THEN SW = 1: GOTO 1555
1550  IF ASC (X$) < 48 OR ASC (X$) > 57 THEN 1515
1555  B$ = B$ + X$: PRINT X$
1560  IX = IX + 1: GOTO 1505
1565  IF IX = 1 THEN 1505
1570  HTAB H + IX: PRINT " ";: HTAB H + IX - 1
1575  IF MID$ (B$, LEN (B$), 1) = "." THEN SW = 0
1580  B$ = MID$ (B$, 1, LEN (B$) - 1)
1585  IX = IX - 1: GOTO 1505
1590  CALL - 868
1595  VTAB V: HTAB H + 1: PRINT B$
1600  RETURN

```

Figure 10.3. Number filter subroutine.

In order to use the above subroutine, the calling code is as follows:

```
30380 V = 9: H = 21: GOSUB 1500: R = VAL (B$)
```

Line 30380 defines the variables, *V* and *H*, to be used by the subroutine at line 1500. *B\$* is the variable that is returned from the subroutine. *B\$* contains a number, in string form, that can be recovered by taking the VALue of the string.

<i>Line</i>	<i>Description</i>
Line 1500	Subroutine setup. Index set to one. $IX = 1$ N\$ set to the null string. $N\$ = ""$ Switch set to zero. $SW = 0$
Line 1505	Position the cursor. Vertically tab to the V value. $VTAB V$ . Horizontally tab to the H value plus index. $HTAB H + IX$ .
Line 1510	Print the "+" sign in the cursor position. NOTE: You may use the prompt character of your own choosing.
Line 1515	Reposition the cursor on the screen.
Line 1520	Get the first character from the keyboard. GET X\$. NOTE: Lines 1525-1545 constitute the command table of acceptable commands (keystrokes) allowed.
Line 1525	Test for a carriage return. $13 = CR$ .
Line 1530	Test for a backspace character. $8 = BS$ .
Line 1535	Test index for value greater than one.
Line 1540	Test for a minus sign. $45 = "-"$ . Minus sign allowed only in position one.
Line 1545	Test for decimal point. $46 = "."$ . SW also must be a zero. Signifies no decimal point has been entered. Set the switch to one.
Line 1550	Test range of character entered. If character is outside the range of allowed numbers, ignore. Numbers are in the collating sequence of 48 through 57 decimal, which is the range of decimal numbers from 0-9.
Line 1555	Legal character entered. Add character to the output string. $B\$ = B\$ + X\$$ . Print character entered. $PRINT X\$$ .
Line 1560	Increment and loop. Increment index. $IX = IX + 1$ . Loop back for next character. $GOTO 1505$ .
Line 1565	Test IX for a value of one. If true then branch to line 1505.
Line 1570	Reposition cursor and print a blank space.
Line 1575	Test for decimal point in last position of string. If "." then reset switch to zero. $SW = 0$ .
Line 1580	Erase the last character from B\$.
Line 1585	Decrement index. $IX = IX - 1$ . Loop back. $GOTO 1505$ .

Line 1590 Clear current line to the end.  
 Line 1595 Reposition cursor and print B\$.  
 Line 1600 RETURN from subroutine.

The string filter routine below allows a much broader range of characters to be entered. This is to accommodate the entry of a name, address, etc. Type in this routine and save it on your library diskette.

```

1395 REM * STRING FILTER
1400 IX = 1: B$ = ""
1405 VTAB V: HTAB H + IX
1410 PRINT "*"
1415 VTAB V: HTAB H + IX
1420 GET X$
1422 IF ASC (X$) = 27 THEN 1490
1425 IF ASC (X$) = 13 THEN 1480
1430 IF ASC (X$) = 8 THEN 1460
1435 IF ASC (X$) = 32 THEN 1450
1440 IF ASC (X$) > 45 AND ASC (X$) < 58 THEN 1450
1442 IF ASC (X$) > 57 AND ASC (X$) < 65 THEN 1415
1445 IF ASC (X$) < 65 OR ASC (X$) > 90 THEN 1415
1450 B$ = B$ + X$: PRINT X$
1455 IX = IX + 1: GOTO 1405
1460 IF IX = 1 THEN 1405
1465 HTAB H + IX: PRINT " "; HTAB H + IX - 1
1470 B$ = MID$ (B$, 1, LEN (B$) - 1)
1475 IX = IX - 1: GOTO 1405
1480 CALL - 868
1485 VTAB V: HTAB H + 1: PRINT B$
1490 RETURN

```

Figure 10.4. String filter subroutine.

The driving code for the above subroutine is as follows:

```

30520 V = 8 + N: H = 13: GOSUB 1400
30530 L$(N) = B$

```

The V sets the vertical position of the cursor from the top of the screen. The H sets the horizontal position of the cursor from the left edge of a line. The subroutine is called with GOSUB 1400. In general, this routine is the same as Figure 10.3, but this routine allows a much broader list of characters to be accepted into the program. The list of acceptable, allowed characters (command table) is shown in lines 1425–1445. If you have an Apple IIe, you may want to allow for lowercase characters. If so, you will need to add to the range of allowed characters.

This subroutine is used to enter names, addresses, etc., into a program and at the same time eliminate special characters, control characters, and the like. You may want to enter this routine separately and save it as `STRING.FILTER` and then merge it into `XMAS.PROG`.

You should be able to analyze this routine to determine what it accomplishes. If you do not know the ASCII character set, we refer you to pages 138 and 139 of the Applesoft II BASIC manual or page 241 to 244 in the Apple IIe reference manual or Appendix I of this book. From the approach taken in Figures 10.3 and 10.4, you will be able to test each character as it is entered and before it is assigned to the final output string. In this manner, you can create any number of routines to be used for specific purposes just by changing the allowable character set. You should find a number of applications for these routines or their modifications.

### 10.2.3. Pathname determination subroutine

Since ProDOS has the capability to find diskette files through the use of pathnames only, it is extremely important that you have a correct file pathname and file name. With that in mind, it is necessary that you determine the pathname and file name correctly. The following code should help you do just that. In this case, the program must be located within the diskette directory. If you locate this program within another a subdirectory, the code will have to be modified to accommodate that structure.

```

5000  VTAB 7: CALL - 958:SW = 0: PRINT "Is PREFIX correct?
      (Y/N) ";: GET Q$: PRINT Q$
5010  IF ASC ( LEFT$ (Q$,1)) = 27 THEN SW = 1: GOTO 5295
5020  IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
      5100
5030  IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
      5200
5040  GOSUB 510: GOTO 5000
5100  VTAB 9: CALL - 958: PRINT "New file name = ";:V = 9:
      H = 16: GOSUB 1400:L = LEN (B$)
5102  IF ASC (X$) = 27 THEN PRINT : GOTO 5000
5105  IF L > 15 THEN GOSUB 510: GOTO 5100
5110  F$ = B$:PN$ = PR$ + F$
5120  VTAB 7: CALL - 958: PRINT "PATHNAME = ";PN$
5130  VTAB 9: CALL - 958: GOSUB 550
5135  IF ASC ( LEFT$ (Q$,1)) = 27 THEN 5000
5140  IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
      5295
5150  IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
      5000
5160  GOSUB 510: GOTO 5130
5200  VTAB 9: CALL - 958: PRINT "Enter new PREFIX = ";:V =
      9:H = 19: GOSUB 1400
5205  IF ASC (X$) = 27 THEN PRINT : GOTO 5000
5210  IF LEFT$ (B$,1) < > "/" THEN B$ = "/" + B$

```

```

5220 L = LEN (B$): IF L > 15 THEN GOSUB 510: GOTO 5200
5230 IF RIGHT$ (B$,1) < > "/" THEN B$ = B$ + "/"
5240 VTAB 5: GOSUB 8950:PR$ = B$:V = 5: GOSUB 540
5250 GOTO 5000
5295 RETURN

```

Figure 10.5. Pathname subroutine.

Following is a description of the above code.

<i>Line</i>	<i>Description</i>
Line 5000	Prompt operator for the correct PREFIX.
Line 5010	Test last character entered for an ESC key entry. If ESC key was entered then branch to line 5295.
Line 5020	Test for a Y answer. If true branch to line 5100.
Line 5030	Test for an N answer. If true branch to line 5200.
Line 5040	Activate the general error routine and then go to line 5000.
Line 5100	Enter the new file name.
Line 5102	Test last character entered for an ESC key entry. If ESC key was entered then branch to line 5000.
Line 5105	Test length of entry. If greater than 15 characters then do not accept.
Line 5110	Make assignment to the file name variable, F\$. Make assignment to the pathname variable, PN\$.
Line 5120	Enter the new pathname.
Line 5130	Position cursor and ask operator to validate correctness of entry.
Line 5135	Test last character entered for an ESC key entry. If ESC key was entered then branch to line 5000.
Line 5140	Test for a Y answer. If true then set the RF variable to 1 and go to line 5100.
Line 5150	Test for an N answer. If true then go to line 5200.
Line 5160	Activate general error message and then branch to line 5130.
Line 5200	Prompt operator for the entry of a new prefix name.
Line 5205	Test last character entered for an ESC key entry. If ESC key was entered then branch to line 5000.
Line 5210	Test for the entry of a "/" at the beginning of the partial pathname. If not entered then add "/" to the pathname.
Line 5220	Test length of pathname entry. If length is greater than 15 characters, entry will not be accepted.

Line 5230    Test for the entry of a "/" at the end of partial pathname.  
               If not entered then add "/" to pathname.  
 Line 5240    Print a row of asterisks.  
               Print the current prefix on the video screen.  
 Line 5250    Make a hard branch to line 5000.  
 Line 5295    Return from subroutine.

#### 10.2.4. Print to screen subroutine

The following small code section is for the purpose of printing to the screen the data elements needed to be entered for each record in the file. This routine is used in a number of major subroutines.

```
2999  REM * PRINT SCREEN
3000  PRINT "1- NAME      :"
3010  PRINT "2- ADDRESS  :"
3020  PRINT "3- CITY     :"
3030  PRINT "4- STATE    :"
3040  PRINT "5- ZIP CODE  :"
3050  PRINT "6- AREA CODE:"
3060  PRINT "7- PHONE #   :"
3070  RETURN
```

Figure 10.6. Screen data entry.

The description of the code is shown below.

<i>Line</i>	<i>Description</i>
Line 3000	Print NAME on screen.
Line 3010	Print ADDRESS on screen.
Line 3020	Print CITY on screen.
Line 3030	Print STATE on screen.
Line 3040	Print ZIP CODE on screen.
Line 3050	Print AREA CODE on screen.
Line 3060	Print PHONE # on screen.
Line 3070	RETURN from subroutine.

#### 10.2.5. Read a record subroutines

The following two subroutines are for the purpose of reading any record in the file and reading record zero.

```

1999 REM * READ RECORD R
2000 PRINT D$;"OPEN ";PN$;" ,L128,S";SY(2);",D";SY(4)
2010 PRINT D$;"READ ";PN$;" ,R";R
2020 INPUT L$(1),L$(2),L$(3),L$(4),L$(5),L$(6),L$(7)
2025 INPUT YR$(0),YR$(1),YR$(2),YR$(3),YR$(4),YR$(5),YR$(6
    ),YR$(7),YR$(8),YR$(9)
2030 PRINT D$;"CLOSE"
2040 RETURN

2099 REM * READ RECORD 0
2100 PRINT D$;"OPEN ";PN$;" ,L128,S";SY(2);",D";SY(4)
2110 PRINT D$;"READ ";PN$;" ,R0"
2120 INPUT TR,NR
2130 PRINT D$;"CLOSE"
2140 RETURN

```

Figure 10.7. Read record subroutines.

<i>Line</i>	<i>Description</i>
Line 2000	Open communications with the file PN\$. Record length is 128 bytes. File is located in slot SY(2) on drive SY(4).
Line 2010	Communications is to read data from record R.
Line 2020	Input the name and address data.
Line 2025	Input the year data.
Line 2030	Close communications with the disk operating system.
Line 2040	Return from the subroutine.
Line 2100	Open communications with the file PN\$. Record length is 128 bytes File is located in slot SY(2) on drive SY(4).
Line 2110	Read record zero from the file.
Line 2120	Input the variables TR and NR from the diskette.
Line 2130	Close communication with the disk operating system.
Line 2140	Return from the subroutine.

### 10.2.6. Write a record subroutines

The following two subroutines are for the purpose of writing to any record in the file and writing to record zero.

```

2499 REM * WRITE RECORD R
2500 PRINT D$;"OPEN ";PN$;" ,L128,S";SY(2);",D";SY(4)
2510 PRINT D$;"WRITE ";PN$;" ,R";R
2520 FOR I = 1 TO 7: PRINT L$(I): NEXT I
2525 FOR I = 0 TO 9: PRINT YR$(I): NEXT I
2530 PRINT D$;"CLOSE"
2540 RETURN

2599 REM * WRITE RECORD 0
2600 PRINT D$;"OPEN ";PN$;" ,L128,S";SY(2);",D";SY(4)
2610 PRINT D$;"WRITE ";PN$;" ,R0"
2620 PRINT TR: PRINT NR
2630 PRINT D$;"CLOSE"
2640 RETURN

```

Figure 10.8. Write record subroutine.

A description of what each line of code accomplishes is shown below.

<i>Line</i>	<i>Description</i>
Line 2500	Open communications with the file whose pathname is defined by PN\$. The record length is 128 bytes. The file is located on the diskette in slot SY(2), drive SY(4).
Line 2510	Communications with the diskette is to write to the file. Routine will write to record number R. Note: It is necessary that you define the record to be written before calling this subroutine.
Line 2520	Print all of the name and address information to the diskette.
Line 2525	Print all of the year data to the diskette.
Line 2530	Close communication with the disk operating system.
Line 2540	Return from subroutine.
Line 2600	Open communications with the file whose name is defined by PN\$. Record length is 128 bytes. The file is located in slot SY(2) on drive SY(4).
Line 2610	The communications is to write to the file. Record 0 will be written.
Line 2620	Print the two record pointers, TR and NR.
Line 2630	Close communications with the disk operating system.
Line 2640	Return from subroutine.

### 10.2.7. System configuration setup subroutine

The following code is for the purpose of prompting the operator to either accept the default values or entering your own configuration for the items specified.



```

3100 PRINT "DISKS:"
3110 PRINT TAB( 6)"BOOT SLOT:"
3120 PRINT TAB( 6)"DATA SLOT:"
3130 PRINT
3140 PRINT TAB( 5)"BOOT DRIVE:"
3150 PRINT TAB( 5)"DATA DRIVE:"
3160 PRINT
3170 PRINT "PRINTER:"
3180 PRINT TAB( 11)"SLOT:"
3190 PRINT TAB( 11)"TYPE:"
3200 VTAB 15: HTAB 25: PRINT "1=SERIAL"
3210 VTAB 16: HTAB 25: PRINT "2=PARALLEL"
3250 RETURN

```

Figure 10.9. System configuration subroutine.

The system configuration code is described below.

<i>Line</i>	<i>Description</i>
Line 3100	Print the subtitle on the screen.
Line 3110	Print the boot slot prompt.
Line 3120	Print the data slot prompt.
Line 3130	Print a blank line.
Line 3140	Print the boot drive prompt.
Line 3150	Print the data drive prompt.
Line 3160	Print a blank line.
Line 3170	Print the printer subtitle.
Line 3180	Print the slot prompt.
Line 3190	Print the printer type prompt.
Line 3200	Print the type number for a serial printer.
Line 3210	Print the type number for a parallel printer.
Line 3250	Return from the subroutine.

#### 10.2.8. Thunderclock routine

The following three lines of code are for the purpose of reading the current time and date from the Thunderclock installed in slot 4. Slot 4 is the recommended slot for any clock/calendar card.

```

60 PRINT D$;"IN#4"
62 INPUT "%";T$
64 PRINT D$;"IN#0"

```

Figure 10.10. Thunderclock routine.

A line-by-line description of the above code is shown below.

<i>Line</i>	<i>Description</i>
-------------	--------------------

Line 60	Activate slot 4.
Line 62	Input date/time.
Line 64	Deactivate clock/calendar slot.

**10.2.9. Read current prefix routine**

The following two lines of code are for the purpose of capturing the current prefix.

```
400 PRINT D$;"PREFIX"
405 INPUT PR$
```

Figure 10.11. Read prefix routine.

A description of this code is shown below.

<i>Line</i>	<i>Description</i>
-------------	--------------------

Line 400	Access the prefix on the boot drive.
Line 405	Capture the prefix into the variable PR\$.

**10.2.10. Report heading subroutine**

The following code is for the purpose of providing each printed page a heading so that the report looks reasonably nice.

```
8899 REM * REPORT HEADING
8900 POKE 36,60: PRINT "Page: ";PG
8905 POKE 36,34: PRINT "List Report"
8907 POKE 36,5: PRINT "Name & Address";
8910 POKE 36,60: PRINT "Date: "; LEFT$(T$,5)
8915 FOR I = 1 TO 79: PRINT "=";: NEXT I: PRINT " "
8920 PG = PG + 1:LN = 4
8925 RETURN
```

Figure 10.12. Report heading subroutine.

Below is a line-by-line description of the above code.

<i>Line</i>	<i>Description</i>
-------------	--------------------

Line 8900	Move cursor to position 60. POKE 36,60. Print page number. Note: The instruction POKE 36,x is recommended since TAB, SPC, and HTAB may not work correctly.
-----------	--

- Line 8905 Move cursor to position 34 on paper.  
Print report title on paper.
- Line 8907 Move cursor to position 5 on paper.  
Print "Name & Address".
- Line 8910 Move cursor to position 60 on paper.  
Print the date.
- Line 8915 Print an equal sign, "=", across paper.
- Line 8920 Increment page number.  
Set the line count to 4. LN = 4.
- Notes: (1) The main subroutine that called this subroutine counts lines that are printed so that the next heading will be placed at the proper location on the next page.  
(2) Each page is numbered sequentially. Therefore the subroutine keeps track of the next page number that needs to be printed.
- Line 8925 Return from subroutine.

```

*** XMAS LIST ***

Main menu:                               Date: 01/30
-----
1 -System set-up
2 -Create a file
3 -Delete a file
4 -Add a record
5 -Edit a record
6 -Delete record
7 -Review record
8 -Prints report
9 -Prints labels
0 -Exit program

Which one? ==> *
```

### 10.3. THE PROGRAM

This section is devoted to giving you a complete listing of the program that has been discussed in the previous sections of this chapter. Now would be a good time to power up your system

and then load the program, XMAS.PROG. Then fill in the missing code from what you have previously entered.

This program was written to operate with an Apple IIe with ProDOS installed. If you are operating an Apple II Plus without lowercase capability then change the lowercase character strings to uppercase.

The program is called XMAS.PROG and I used a file name of XMAS.LIST. The total pathname is /SCRATCH.DISK/XMAS.LIST for that file.

```
]NEW
```

```
]LIST
```

```
1  REM ***** XMAS.PROG *****
2  REM *
3  REM * WRITTEN BY:JL CAMPBELL
4  REM *      DATED:12/10/1983
5  REM *
7  REM * COMPUTER:AII+ OR AIIe
8  REM *
9  REM *****
10 TEXT : CLEAR : HOME : POKE 216,0:D$ = CHR$(4)
15 PRINT D$;"FRE"
20 DIM M$(10),TI$(5),TI(5),SL$(7),SL(7)
30 DIM L$(10),YR$(10),CD$(10),SY(10)
60 PRINT D$;"IN#4"
62 INPUT "%";T$
64 PRINT D$;"IN#0"
70 FOR I = 0 TO 7:SL$(I) = "EMPTY":SL(I) = 0: NEXT I
80 FOR I = 1 TO 9: READ M$(I): NEXT I
82 DATA 1 -System set-up,2 -Create a file,3 -Delete a
   file
84 DATA 4 -Adds a record,5 -Edit a record,6 -Delete re
   cord
86 DATA 7 -Review record,8 -Prints report,9 -Prints lab
   els
90 FOR I = 1 TO 6: READ SY(I): NEXT I
92 DATA 6,6,1,1,1,2
94 TF = 0:RF = 0
400 PRINT D$;"PREFIX"
405 INPUT PR$
415 F$ = PR$ + "SYSTEM"
420 EP = 11: ONERR GOTO 60000
425 PRINT D$;"OPEN ";F$
430 PRINT D$;"READ ";F$
435 INPUT SY(1),SY(2),SY(3),SY(4),SY(5),SY(6)
440 PRINT D$;"CLOSE"
490 GOSUB 55000
495 GOTO 9000
497 REM *****
498 REM * GEN'L SUBR *
```

```

499 REM *****
500 PRINT "Which one? ==> ";: GET N$: PRINT N$:N = VAL (N$):NA =
ASC (
    N$): RETURN
510 VTAB 22: CALL - 958: HTAB 13: INVERSE : PRINT "ILLEGAL
    ENTRY!"; CHR$ (7): NORMAL
520 FOR L = 1 TO 1000: NEXT L: RETURN
530 VTAB V: HTAB H: PRINT "Date: "; LEFT$ (T$,5): RETURN
540 VTAB V: PRINT "*CURRENT PREFIX = ";PR$: RETURN
550 PRINT "Is that correct? (Y/N) ";: GET Q$: PRINT Q$: RETURN
560 PRINT "Press any key to continue...";: GET Q$: PRINT :
    RETURN
570 VTAB V: PRINT "*ESC=ESCAPE": RETURN
580 VTAB V: PRINT "DO YOU WANT THAT? (Y/N) ": RETURN
590 PRINT "Any changes? (Y/N) ";: GET Q$: PRINT Q$: RETURN
600 PRINT "Another entry? (Y/N) ";: GET Q$: PRINT Q$:
    RETURN
1395 REM * STRING FILTER
1400 IX = 1:B$ = ""
1405 VTAB V: HTAB H + IX
1410 PRINT "*"
1415 VTAB V: HTAB H + IX
1420 GET X$
1422 IF ASC (X$) = 27 THEN 1490
1425 IF ASC (X$) = 13 THEN 1480
1430 IF ASC (X$) = 8 THEN 1460
1435 IF ASC (X$) = 32 THEN 1450
1440 IF ASC (X$) > 45 AND ASC (X$) < 58 THEN 1450
1442 IF ASC (X$) > 57 AND ASC (X$) < 65 THEN 1415
1445 IF ASC (X$) < 65 OR ASC (X$) > 90 THEN 1415
1450 B$ = B$ + X$: PRINT X$
1455 IX = IX + 1: GOTO 1405
1460 IF IX = 1 THEN 1405
1465 HTAB H + IX: PRINT " ";: HTAB H + IX - 1
1470 B$ = MID$ (B$,1, LEN (B$) - 1)
1475 IX = IX - 1: GOTO 1405
1480 CALL - 868
1485 VTAB V: HTAB H + 1: PRINT B$
1490 RETURN
1495 REM * NUMBER FILTER
1500 IX = 1:B$ = "":SW = 0
1505 VTAB V: HTAB H + IX
1510 PRINT "+";
1515 VTAB V: HTAB H + IX
1520 GET X$
1525 IF ASC (X$) = 13 THEN 1590
1530 IF ASC (X$) = 8 THEN 1565
1535 IF IX > 1 THEN 1545
1540 IF ASC (X$) = 45 THEN 1555
1545 IF ASC (X$) = 46 AND SW = 0 THEN SW = 1: GOTO 1555
1550 IF ASC (X$) < 48 OR ASC (X$) > 57 THEN 1515
1555 B$ = B$ + X$: PRINT X$
1560 IX = IX + 1: GOTO 1505

```

```

1565 IF IX = 1 THEN 1505
1570 HTAB H + IX: PRINT " ";: HTAB H + IX - 1
1575 IF MID$(B$, LEN(B$), 1) = "." THEN SW = 0
1580 B$ = MID$(B$, 1, LEN(B$) - 1)
1585 IX = IX - 1: GOTO 1505
1590 CALL - 868
1595 VTAB V: HTAB H + 1: PRINT B$
1600 RETURN
1999 REM * READ RECORD R
2000 PRINT D$;"OPEN ";PN$;" ,L128,S";SY(2);",D";SY(4)
2010 PRINT D$;"READ ";PN$;" ,R";R
2020 INPUT L$(1),L$(2),L$(3),L$(4),L$(5),L$(6),L$(7)
2025 INPUT YR$(0),YR$(1),YR$(2),YR$(3),YR$(4),YR$(5),YR$(6)
      ,YR$(7),YR$(8),YR$(9)
2030 PRINT D$;"CLOSE"
2040 RETURN
2099 REM * READ RECORD O
2100 PRINT D$;"OPEN ";PN$;" ,L128,S";SY(2);",D";SY(4)
2110 PRINT D$;"READ ";PN$;" ,RO"
2120 INPUT TR,NR
2130 PRINT D$;"CLOSE"
2140 RETURN
2499 REM * WRITE RECORD R
2500 PRINT D$;"OPEN ";PN$;" ,L128,S";SY(2);",D";SY(4)
2510 PRINT D$;"WRITE ";PN$;" ,R";R
2520 FOR I = 1 TO 7: PRINT L$(I): NEXT I
2525 FOR I = 0 TO 9: PRINT YR$(I): NEXT I
2530 PRINT D$;"CLOSE"
2540 RETURN
2599 REM * WRITE RECORD O
2600 PRINT D$;"OPEN ";PN$;" ,L128,S";SY(2);",D";SY(4)
2610 PRINT D$;"WRITE ";PN$;" ,RO"
2620 PRINT TR: PRINT NR
2630 PRINT D$;"CLOSE"
2640 RETURN
2999 REM * PRINT SCREEN
3000 PRINT "1- NAME      : "
3010 PRINT "2- ADDRESS   : "
3020 PRINT "3- CITY       : "
3030 PRINT "4- STATE        : "
3040 PRINT "5- ZIP CODE      : "
3050 PRINT "6- AREA CODE    : "
3060 PRINT "7- PHONE #      : "
3070 RETURN
3100 PRINT "DISKS:"
3110 PRINT TAB( 6)"BOOT SLOT:"
3120 PRINT TAB( 6)"DATA SLOT:"
3130 PRINT
3140 PRINT TAB( 5)"BOOT DRIVE:"
3150 PRINT TAB( 5)"DATA DRIVE:"
3160 PRINT
3170 PRINT "PRINTER:"
3180 PRINT TAB( 11)"SLOT:"

```

```

3190 PRINT TAB( 11)"TYPE:"
3200 VTAB 15: HTAB 25: PRINT "1=SERIAL"
3210 VTAB 16: HTAB 25: PRINT "2=PARALLEL"
3250 RETURN
5000 VTAB 7: CALL - 958:SW = 0: PRINT "Is PREFIX correct?
      (Y/N) ";: GET Q$: PRINT Q$
5010 IF ASC ( LEFT$ (Q$,1)) = 27 THEN SW = 1: GOTO 5295
5020 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
      5100
5030 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
      5200
5040 GOSUB 510: GOTO 5000
5100 VTAB 9: CALL - 958: PRINT "New file name = ";:V = 9:
      H = 16: GOSUB 1400:L = LEN (B$)
5102 IF ASC (X$) = 27 THEN PRINT : GOTO 5000
5105 IF L > 15 THEN GOSUB 510: GOTO 5100
5110 F$ = B$:PN$ = PR$ + F$
5120 VTAB 7: CALL - 958: PRINT "PATHNAME = ";PN$
5130 VTAB 9: CALL - 958: GOSUB 550
5135 IF ASC ( LEFT$ (Q$,1)) = 27 THEN 5000
5140 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
      RF = 1:GOTO 5295
5150 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
      5000
5160 GOSUB 510: GOTO 5130
5200 VTAB 9: CALL - 958: PRINT "Enter new PREFIX = ";:V =
      9:H = 19: GOSUB 1400
5205 IF ASC (X$) = 27 THEN PRINT : GOTO 5000
5210 IF LEFT$ (B$,1) < > "/" THEN B$ = "/" + B$
5220 L = LEN (B$): IF L > 15 THEN GOSUB 510: GOTO 5200
5230 IF RIGHT$ (B$,1) < > "/" THEN B$ = B$ + "/"
5240 VTAB 5: GOSUB 8950:PR$ = B$:V = 5: GOSUB 540
5250 GOTO 5000
5295 RETURN
5299 REM * PRINTER SET-UP
5300 VTAB 9: CALL - 958
5310 PRINT "Do the following:"
5320 PRINT TAB( 17)"1- LOAD printer paper"
5330 PRINT TAB( 17)"2- Set TOP-OF-FORM"
5340 PRINT TAB( 17)"3- Turn ON printer"
5350 PRINT TAB( 17)"4- Printer ON-LINE"
5360 RETURN
8899 REM * REPORT HEADING
8900 POKE 36,60: PRINT "Page: ";PG
8905 POKE 36,34: PRINT "List Report"
8907 POKE 36,5: PRINT "Name & Address";
8910 POKE 36,60: PRINT "Date: "; LEFT$ (T$,5)
8915 FOR L = 1 TO 79: PRINT "=";: NEXT L: PRINT " "
8920 PG = PG + 1:LN = 4
8925 RETURN
8950 FOR I = 1 TO 39: PRINT "*";: NEXT I: PRINT : RETURN
8960 PRINT "*";: HTAB 39: PRINT "*": RETURN
8970 FOR I = 1 TO 39: PRINT "-";: NEXT I: PRINT : RETURN

```

```

8999 REM * MAIN DRIVER
9000 HOME : PRINT : PRINT TAB( 11)"*** XMAS LIST ***":
      PRINT
9010 VTAB 4: PRINT "Main menu:";: HTAB 29: PRINT "Date: ";
      LEFT$ (T$,5): PRINT "---- ----";: HTAB 29: PRINT "--
      --- ----"
9015 FOR I = 1 TO 9: VTAB 5 + I: PRINT TAB( 11)M$(I): NEXT
      I
9020 VTAB 16: PRINT TAB( 11)"0 -Exit program"
9025 IF TF = 1 THEN VTAB 6: PRINT "DO THIS"
9030 VTAB 18: CALL - 958: GOSUB 500
9040 IF N > 0 AND N < 10 THEN 9100
9050 IF NA = 48 THEN 9990
9060 GOSUB 510: GOTO 9030
9100 VTAB 5 + N: HTAB 5: PRINT " ==> ";: PRINT M$(N);" <
      =="
9110 VTAB 20: CALL - 958: PRINT QA$;: GET Q$: PRINT Q$
9120 IF Q$ = "Y" OR Q$ = "y" THEN 9200
9130 IF Q$ = "N" OR Q$ = "n" THEN 9150
9140 GOSUB 510: GOTO 9110
9150 VTAB 5 + N: PRINT " ";M$(N);" "
9160 GOTO 9030
9200 ON N GOSUB 10000,15000,20000,25000,30000,35000,40000,
      45000,50000
9210 GOTO 9000
9990 HOME : VTAB 10: PRINT "GOODBYE FOR NOW..."
9995 END
10000 HOME : GOSUB 8950: GOSUB 8960
10010 PRINT "*";: HTAB 13: PRINT "SYSTEM SET-UP";: HTAB 39
      : PRINT "*"
10020 GOSUB 8960: GOSUB 8950
10030 V = 1:H = 28: GOSUB 530
10040 V = 5: GOSUB 540
10050 V = 1: GOSUB 570
10055 IF RF = 1 THEN 10100
10060 GOSUB 5000
10070 IF SW = 1 THEN 14995
10100 VTAB 1: PRINT "*RETURN=ACCEPT"
10110 VTAB 7: CALL - 958
10120 GOSUB 3100
10199 REM * ID SYSTEM
10200 FOR I = 1 TO 2: VTAB 7 + I: HTAB 17
10210 PRINT SY(I)
10220 VTAB 7 + I: HTAB 17: GET X$
10230 IF ASC (X$) = 13 THEN PRINT SY(I): GOTO 10250
10232 IF VAL (X$) > 0 AND VAL (X$) < 8 THEN 10240
10235 GOSUB 510: GOTO 10220
10240 SY(I) = VAL (X$): PRINT SY(I)
10250 NEXT I
10300 FOR I = 3 TO 4: VTAB 8 + I: HTAB 17
10310 PRINT SY(I)
10320 VTAB 8 + I: HTAB 17: GET X$
10330 IF ASC (X$) = 13 THEN PRINT SY(I): GOTO 10350

```



```

10332 IF VAL (X$) > 0 AND VAL (X$) < 8 THEN 10340
10335 GOSUB 510: GOTO 10320
10340 SY(I) = VAL (X$): PRINT SY(I)
10350 NEXT I
10400 FOR I = 5 TO 6: VTAB 10 + I: HTAB 17
10410 PRINT SY(I)
10420 VTAB 10 + I: HTAB 17: GET X$
10430 IF ASC (X$) = 13 THEN PRINT SY(I): GOTO 10450
10432 IF VAL (X$) = 1 OR VAL (X$) = 2 THEN 10440
10435 GOSUB 510: GOTO 10420
10440 SY(I) = VAL (X$): PRINT SY(I)
10450 NEXT I
10500 VTAB 18: CALL - 958: GOSUB 590
10510 IF ASC ( LEFT$ (Q$,1) ) = 27 THEN 14995
10520 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
10110
10530 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
10600
10540 GOSUB 510: GOTO 10500
10600 VTAB 18: CALL - 958: PRINT "Saving system configura
tion."
10610 EP = 10: ONERR GOTO 60000
10615 F$ = PR$ + "SYSTEM"
10620 PRINT D$;"CREATE ";F$;" ,TTXT,S";SY(1);",D";SY(3)
10630 PRINT D$;"DELETE ";F$;" ,S";SY(1);",D";SY(3)
10640 PRINT D$;"OPEN ";F$;" ,S";SY(1);",D";SY(3)
10650 PRINT D$;"WRITE ";F$
10660 FOR I = 1 TO 6
10670 PRINT SY(I)
10680 NEXT I
10690 PRINT D$;"CLOSE"
10700 TF = 0
14995 SW = 0: RETURN
15000 HOME : GOSUB 8950: GOSUB 8960
15010 PRINT "*";: HTAB 13: PRINT "CREATE A FILE";: HTAB 39
: PRINT "*"
15020 GOSUB 8960: GOSUB 8950
15030 V = 1:H = 28: GOSUB 530
15040 V = 5: GOSUB 540
15050 V = 1: GOSUB 570
15060 GOSUB 5000
15070 IF SW = 1 THEN 19995
15300 VTAB 9: CALL - 958
15310 SW = 0:EP = 1: ONERR GOTO 60000
15330 TR = 0:NR = 1
15340 PRINT D$;"CREATE ";PN$;" ,TTXT"
15360 PRINT D$;"DELETE ";PN$
15380 GOSUB 2600
15420 POKE 216,0
15430 IF SW = 0 THEN 19995
15500 SW = 0: GOTO 9000
19995 SW = 0: RETURN
20000 HOME : GOSUB 8950: GOSUB 8960

```

```

20010 PRINT "*";: HTAB 13: PRINT "DELETE A FILE";: HTAB 39
      : PRINT "*"
20020 GOSUB 8960: GOSUB 8950
20030 V = 1:H = 28: GOSUB 530
20040 V = 5: GOSUB 540
20050 V = 1: GOSUB 570
20055 IF RF = 1 THEN 20100
20060 GOSUB 5000
20070 IF SW = 1 THEN 24995
20100 VTAB 9: CALL - 958
20110 PRINT "Do you REALLY want to delete "
20120 VTAB 11: PRINT PN$
20130 VTAB 13: CALL - 958
20140 PRINT "Answer with (Y/N). ";: GET Q$: PRINT Q$
20150 IF ASC ( LEFT$ (Q$,1)) = 27 THEN 20000
20160 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
      20200
20170 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
      24995
20180 GOSUB 510: GOTO 20140
20200 PRINT D$;"DELETE ";PN$
24995 SW = 0: RETURN
25000 HOME : GOSUB 8950: GOSUB 8960
25010 PRINT "*";: HTAB 13: PRINT "ADDS A RECORD";: HTAB 39: PRINT "*"
25020 GOSUB 8960: GOSUB 8950
25030 V = 1:H = 28: GOSUB 530
25040 V = 5: GOSUB 540
25050 V = 1: GOSUB 570
25055 IF RF = 1 THEN 25300
25060 GOSUB 5000
25070 IF SW = 1 THEN 29995
25300 VTAB 7: CALL - 958
25305 EP = 3; ONERR GOTO 60000
25310 GOSUB 2100
25315 POKE 216,0
25350 L = LEN (F$):H = 36 - L
25360 PRINT "--> RECORD:";NR;: HTAB H: PRINT F$;" <--"
25370 VTAB 9: CALL - 958: GOSUB 3000
25380 FOR I = 1 TO 7:SW = 0
25390 V = 8 + I:H = 13: GOSUB 1400
25395 IF ASC (X$) = 27 THEN I = 8:SW = 1
25400 L$(I) = B$
25410 NEXT I
25415 IF SW = 1 THEN PRINT : GOTO 25060
25420 VTAB 17: CALL - 958: GOSUB 590
25425 IF ASC ( LEFT$ (Q$,1)) = 27 THEN 25300
25430 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
      25500
25440 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
      25600
25450 GOSUB 510: GOTO 25420
25500 VTAB 17: CALL - 958: GOSUB 500
25502 IF NA = 27 THEN 25420

```

```

25505 IF N > 0 AND N < 8 THEN 25520
25510 GOSUB 510: GOTO 25500
25520 V = N + 8: H = 13: GOSUB 1400
25530 L$(N) = B$
25540 GOTO 25420
25600 VTAB 17: CALL - 958
25610 FOR I = 0 TO 9: HTAB I * 4 + 1
25620 PRINT I + 80;: NEXT I: PRINT
25630 FOR I = 0 TO 9: VTAB 18: HTAB I * 4 + 1: PRINT "N";:
    YR$(I) = "N": NEXT I: PRINT
25700 I = 0
25710 VTAB 18: HTAB I * 4 + 1
25720 GET X$
25722 IF ASC (X$) = 27 THEN PRINT : GOTO 25420
25725 IF ASC (X$) = 13 THEN 25840
25730 IF ASC (X$) = 21 OR ASC (X$) = 10 THEN 25800
25740 IF ASC (X$) = 8 OR ASC (X$) = 11 THEN 25810
25750 IF ASC (X$) = 89 THEN 25790
25760 GOTO 25710
25790 YR$(I) = "Y": PRINT "Y": I = I + 1: IF I > 9 THEN I =
    0
25795 GOTO 25710
25800 I = I + 1: IF I > 9 THEN I = 0
25805 GOTO 25710
25810 I = I - 1: IF I < 0 THEN I = 9
25815 GOTO 25710
25840 PRINT
25850 VTAB 19: CALL - 958: GOSUB 590
25855 IF ASC ( LEFT$ (Q$,1)) = 27 THEN 25700
25860 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN VTAB
    19: CALL - 958: GOTO 25700
25870 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
    25900
25880 GOSUB 510: GOTO 25850
25900 VTAB 19: CALL - 958
25905 EP = 2: ONERR GOTO 60000
25910 R = NR: GOSUB 2500
25915 POKE 216,0
25920 EP = 3: ONERR GOTO 60000
25925 TR = NR: NR = NR + 1: GOSUB 2600
25930 POKE 216,0
25950 VTAB 20: CALL - 958: GOSUB 600
25955 IF ASC ( LEFT$ (Q$,1)) = 27 THEN 25060
25960 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
    25300
25965 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
    29995
25970 GOSUB 510: GOTO 25950
29995 SW = 0: RETURN
30000 HOME : GOSUB 8950: GOSUB 8960
30010 PRINT "*";: HTAB 13: PRINT "EDIT A RECORD";: HTAB 39
    : PRINT "*"
30020 GOSUB 8960: GOSUB 8950

```

```

30030 V = 1:H = 28: GOSUB 530
30040 V = 5: GOSUB 540
30050 V = 1: GOSUB 570
30055 IF RF = 1 THEN 30300
30060 GOSUB 5000
30070 IF SW = 1 THEN 34995
30300 VTAB 7: CALL - 958
30305 EP = 4: ONERR GOTO 60000
30310 GOSUB 2100
30315 POKE 216,0
30350 L = LEN (F$):H = 36 - L
30360 PRINT "--> RECORD:";TR;: HTAB H: PRINT F$;" <--"
30370 VTAB 9: CALL - 958: PRINT "Edit which record? = "
30380 V = 9:H = 21: GOSUB 1500:R = VAL (B$)
30382 IF ASC (X$) = 27 THEN 30060
30385 IF R > = NR OR R < 1 THEN GOSUB 510: GOTO 30370
30390 L = LEN (F$):H = 36 - L
30395 VTAB 7: CALL - 958: PRINT "--> EDIT:";R;: HTAB H: PRINT
F$;" <--"
30400 GOSUB 2000
30410 VTAB 9: CALL - 958
30420 GOSUB 3000
30430 FOR I = 1 TO 7: VTAB 8 + I: HTAB 14: PRINT L$(I): NEXT
I
30440 VTAB 18: CALL - 958: GOSUB 590
30450 IF ASC ( LEFT$ (Q$,1)) = 27 THEN 30060
30460 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
30500
30470 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
30600
30480 GOSUB 510: GOTO 30440
30500 VTAB 18: CALL - 958: GOSUB 500
30502 IF NA = 27 THEN 30440
30505 IF N > 0 AND N < 8 THEN 30520
30510 GOSUB 510: GOTO 30500
30520 V = N + 8:H = 13: GOSUB 1400
30530 L$(N) = B$
30540 GOTO 30440
30600 VTAB 17: CALL - 958
30610 FOR I = 0 TO 9: HTAB I * 4 + 1
30620 PRINT I + 80;: NEXT I: PRINT
30630 FOR I = 0 TO 9: VTAB 18: HTAB I * 4 + 1: PRINT YR$(I
);: NEXT I: PRINT
30700 I = 0
30710 VTAB 18: HTAB I * 4 + 1
30720 GET X$
30722 IF ASC (X$) = 27 THEN PRINT : GOTO 30440
30725 IF ASC (X$) = 13 THEN 30840
30730 IF ASC (X$) = 21 OR ASC (X$) = 10 THEN 30800
30740 IF ASC (X$) = 8 OR ASC (X$) = 11 THEN 30810
30750 IF ASC (X$) = 89 THEN 30790
30760 IF ASC (X$) = 78 THEN 30780
30770 GOTO 30710

```

```

30780 YR$(I) = "N": PRINT "N":I = I + 1: IF I > 9 THEN I =
0
30785 GOTO 30710
30790 YR$(I) = "Y": PRINT "Y":I = I + 1: IF I > 9 THEN I =
0
30795 GOTO 30710
30800 I = I + 1: IF I > 9 THEN I = 0
30805 GOTO 30710
30810 I = I - 1: IF I < 0 THEN I = 9
30815 GOTO 30710
30840 PRINT
30850 VTAB 19: CALL - 958: GOSUB 590
30855 IF ASC ( LEFT$ (Q$,1)) = 27 THEN 30700
30860 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN VTAB
19: CALL - 958: GOTO 30700
30870 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
30900
30880 GOSUB 510: GOTO 30850
30900 VTAB 19: CALL - 958
30910 GOSUB 2500
30950 VTAB 20: CALL - 958: GOSUB 600
30955 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
30300
30960 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
34995
30970 GOSUB 510: GOTO 30950
34995 SW = 0: RETURN
35000 HOME : GOSUB 8950: GOSUB 8960
35010 PRINT "*";: HTAB 13: PRINT "DELETE RECORD";: HTAB 39
: PRINT "*"
35020 GOSUB 8960: GOSUB 8950
35030 V = 1:H = 28: GOSUB 530
35040 V = 5: GOSUB 540
35050 V = 1: GOSUB 570
35055 IF RF = 1 THEN 35300
35060 GOSUB 5000
35070 IF SW = 1 THEN 39995
35300 VTAB 7: CALL - 958
35305 EP = 5: ONERR GOTO 60000
35310 GOSUB 2100
35315 POKE 216,0
35350 L = LEN (F$):H = 36 - L
35360 VTAB 7: CALL - 958: PRINT "--> RECORD: ";TR;: HTAB H
: PRINT F$;" <--"
35370 VTAB 9: CALL - 958: PRINT "Delete which record? = "
35380 V = 9:H = 23: GOSUB 1500:R = VAL (B$)
35385 IF R > = NR OR R < 1 THEN GOSUB 510: GOTO 35370
35390 L = LEN (F$):H = 36 - L
35395 VTAB 7: CALL - 958: PRINT "--> DELETE: ";R;: HTAB H:
PRINT F$;" <--"
35400 GOSUB 2000
35410 VTAB 9: CALL - 958
35420 GOSUB 3000

```

```

35430 FOR I = 1 TO 7: VTAB 8 + I: HTAB 14: PRINT L$(I): NEXT
      I
35440 VTAB 17: CALL - 958
35450 FOR I = 0 TO 9: HTAB I * 4 + 1
35460 PRINT I + 80;: NEXT I: PRINT
35470 FOR I = 0 TO 9: VTAB 18: HTAB I * 4 + 1: PRINT YR$(I
      );: NEXT I: PRINT
35480 VTAB 20: CALL - 958: PRINT "Delete record? (Y/N) ";
      : GET Q$: PRINT Q$
35490 IF ASC ( LEFT$( Q$,1)) = 27 THEN 35000
35495 IF LEFT$( Q$,1) = "Y" OR LEFT$( Q$,1) = "y" THEN
      35600
35500 IF LEFT$( Q$,1) = "N" OR LEFT$( Q$,1) = "n" THEN
      39995
35510 GOSUB 510: GOTO 35480
35600 L$(1) = "?????????":L$(2) = "?????????":L$(5) = "??
      ???":L$(6) = "???":L$(7) = "??? ????"
35610 VTAB 9: CALL - 958
35620 GOSUB 3000
35630 FOR I = 1 TO 7: VTAB 8 + I: HTAB 14: PRINT L$(I): NEXT
      I
35640 VTAB 17: CALL - 958: PRINT "Deleting record....."
35645 GOSUB 2500
35650 VTAB 19: CALL - 958: GOSUB 600
35660 IF ASC ( LEFT$( Q$,1)) = 27 THEN 35000
35670 IF LEFT$( Q$,1) = "Y" OR LEFT$( Q$,1) = "y" THEN
      35700
35680 IF LEFT$( Q$,1) = "N" OR LEFT$( Q$,1) = "n" THEN
      39995
35690 GOSUB 510: GOTO 35650
39995 SW = 0: RETURN
40000 HOME : GOSUB 8950: GOSUB 8960
40010 PRINT "*";: HTAB 13: PRINT "REVIEW RECORD";: HTAB 39
      : PRINT "*"
40020 GOSUB 8960: GOSUB 8950
40030 V = 1:H = 28: GOSUB 530
40040 V = 5: GOSUB 540
40050 V = 1: GOSUB 570
40055 IF RF = 1 THEN 40300
40060 GOSUB 5000
40070 IF SW = 1 THEN 44995
40300 VTAB 7: CALL - 958
40305 EP = 6: ONERR GOTO 60000
40310 GOSUB 2100
40315 POKE 216,0
40350 L = LEN (F$):H = 36 - L
40360 VTAB 7: CALL - 958: PRINT "--> RECORD: ";TR;: HTAB H
      : PRINT F$; " <--"
40370 VTAB 9: CALL - 958: PRINT "Review which record? = "
40380 V = 9:H = 23: GOSUB 1500:R = VAL (B$)
40385 IF R > = NR OR R < 1 THEN GOSUB 510: GOTO 40370
40390 L = LEN (F$):H = 36 - L

```

```

40395 VTAB 7: CALL - 958: PRINT "--> REVIEW: "; R; : HTAB H:
      PRINT F$; " <--"
40400 GOSUB 2000
40410 VTAB 9: CALL - 958
40420 GOSUB 3000
40430 FOR I = 1 TO 7: VTAB 8 + I: HTAB 14: PRINT L$(I): NEXT
      I
40440 VTAB 17: CALL - 958
40450 FOR I = 0 TO 9: HTAB I * 4 + 1
40460 PRINT I + 80; : NEXT I: PRINT
40470 FOR I = 0 TO 9: VTAB 18: HTAB I * 4 + 1: PRINT YR$(I
      ); : NEXT I: PRINT
40480 VTAB 20: CALL - 958: PRINT "Another record? (Y/N) "
      ; : GET Q$: PRINT Q$
40490 IF ASC ( LEFT$ (Q$,1)) = 27 THEN 40000
40495 IF LEFT$ (Q$,1) = "Y" OR LEFT$ (Q$,1) = "y" THEN
      40370
40500 IF LEFT$ (Q$,1) = "N" OR LEFT$ (Q$,1) = "n" THEN
      44995
40510 GOSUB 510: GOTO 40480
44995 SW = 0: RETURN
45000 HOME : GOSUB 8950: GOSUB 8960
45010 PRINT "*"; : HTAB 13: PRINT "PRINTS REPORT"; : HTAB 39
      : PRINT "*"
45020 GOSUB 8960: GOSUB 8950
45030 V = 1: H = 28: GOSUB 530
45040 V = 5: GOSUB 540
45050 V = 1: GOSUB 570
45055 IF RF = 1 THEN 45100
45060 GOSUB 5000
45070 IF SW = 1 THEN 49995
45100 GOSUB 5300
45160 VTAB 15: CALL - 958: GOSUB 560
45170 GOSUB 2100
45180 VTAB 17: PRINT "There are "; TR; " records in file."
45200 PRINT D$; "PR#"; SY(5)
45210 IF SY(6) = 2 THEN PRINT CHR$(9); "80N"
45300 PG = 1: GOSUB 8900
45310 FOR I = 1 TO TR
45315 EP = 7: ONERR GOTO 60000
45320 R = I: GOSUB 2000
45325 POKE 216,0
45330 PRINT I; "-"; : POKE 36,5: PRINT L$(1)
45340 POKE 36,5: PRINT L$(2)
45350 POKE 36,5: PRINT L$(3); ", "; L$(4); " "; L$(5);
45360 POKE 36,50: PRINT "Phone: 1+("; L$(6); ") "; L$(7)
45370 PRINT " "
45380 FOR L = 0 TO 9: POKE 36,L * 4 + 1
45390 PRINT L + 80; : NEXT L: PRINT " "
45400 FOR L = 0 TO 9: POKE 36,L * 4 + 1
45410 PRINT YR$(L); : NEXT L: PRINT " "
45420 PRINT " "

```

```

45430 LN = LN + 7
45440 IF LN < 50 THEN 45480
45450 FOR L = LN + 1 TO 66: PRINT " ": NEXT L
45460 GOSUB 8900
45480 NEXT I
45485 FOR L = LN + 1 TO 66: PRINT " ": NEXT L
45490 PRINT CHR$(9);"40N"
45495 PRINT D$;"PR#0"
49995 SW = 0: RETURN
50000 HOME : GOSUB 8950: GOSUB 8960
50010 PRINT "*";: HTAB 13: PRINT "PRINTS LABELS";: HTAB 39
: PRINT "*"
50020 GOSUB 8960: GOSUB 8950
50030 V = 1:H = 28: GOSUB 530
50040 V = 5: GOSUB 540
50050 V = 1: GOSUB 570
50055 IF RF = 1 THEN 50100
50060 GOSUB 5000
50070 IF SW = 1 THEN 54995
50100 GOSUB 5300
50160 VTAB 15: CALL - 958: GOSUB 560
50165 POKE 216,0
50170 GOSUB 2100
50180 VTAB 17: PRINT "There are ";TR;" records in file."
50200 PRINT D$;"PR#";SY(5)
50210 IF SY(6) = 2 THEN PRINT CHR$(9);"80N"
50300 REM * LABELS
50310 FOR I = 1 TO TR
50320 R = I: GOSUB 2000
50325 PRINT " "
50330 POKE 36,5: PRINT L$(1)
50340 POKE 36,5: PRINT L$(2)
50350 POKE 36,5: PRINT L$(3);", ";L$(4);" ";L$(5)
50360 PRINT " "
50370 PRINT " "
50400 NEXT I
50410 IF SY(6) = 2 THEN PRINT CHR$(9);"40N"
50420 PRINT D$;"PR#0"
54995 SW = 0: RETURN
55000 HOME : GOSUB 8950: GOSUB 8960
55005 PRINT "*";: HTAB 13: PRINT "COMPUTER TYPE";: HTAB 39
: PRINT "*"
55010 GOSUB 8960: GOSUB 8950
55012 V = 1:H = 28: GOSUB 530
55014 V = 5: GOSUB 540
55015 ID = PEEK (49048)
55020 REM * ID COMPUTER TYPE
55025 IF ID > = 192 THEN TI$(1) = "APPLE III":ID = ID -
192:TI(1) = 3: GOTO 55060
55030 IF ID > = 128 THEN TI$(1) = "APPLE IIe":ID = ID -
128:TI(1) = 2: GOTO 55060
55040 IF ID > = 64 THEN TI$(1) = "APPLE II+":ID = ID - 64
:TI(1) = 1: GOTO 55060

```



```

55050 TI$(1) = "APPLE II":TI(1) = 0
55060 REM * ID MEMORY SIZE
55070 IF ID > = 48 THEN TI$(2) = "128K":TI(2) = 3:ID = ID
- 48: GOTO 55110
55080 IF ID > = 32 THEN TI$(2) = "64K":TI(2) = 2:ID = ID
- 32: GOTO 55110
55090 IF ID > = 16 THEN TI$(2) = "48K":TI(2) = 1:ID = ID -
16: GOTO 55110
55100 TI$(2) = "UNKN":TI(2) = 0
55110 REM * 80-COL & TCP
55120 IF ID > = 8 THEN TI$(3) = "UNKN":TI(3) = 3:ID = ID -
8: GOTO 55150
55130 IF ID > = 4 THEN TI$(3) = "UNKN":TI(3) = 2:ID = ID -
4: GOTO 55150
55140 IF ID > = 2 THEN TI$(3) = "80-Column card":TI(3) =
1:SL$(3) = TI$(3):SL(3) = TI(3):ID = ID - 2
55150 IF ID > = 1 THEN TI$(4) = "CLOCK":TI(4) = 1:ID = ID
- 1
55160 VTAB 7: CALL - 958: PRINT "You have an ";TI$(1);" w
ith ";TI$(2);"."
55185 REM * ID SLOTS
55190 CD$(0) = "Used":CD$(1) = "Printer":CD$(2) = "Joystick"
:CD$(3) = "I/O card"
55195 CD$(4) = "MODEM":CD$(5) = "Audio card":CD$(6) = "Cloc
k":CD$(7) = "Mass storage"
55200 CD$(8) = "80-Column card":CD$(9) = "Network card"
55210 ID = PEEK (49049):IX = 128
55220 FOR I = 1 TO 7
55230 SL = - 16384 + 256 * I
55240 IF PEEK (SL + 23) = 201 AND PEEK (SL + 55) = 207 AND
PEEK (SL + 76) = 234 THEN SL$(I) = "Silentype":SL(I) =
I: GOTO 55300
55250 IF PEEK (SL) = 8 AND PEEK (SL + 2) = 40 AND PEEK
(SL + 4) = 88 AND PEEK (SL + 6) = 112 THEN SL$(I) = "
Clock":SL(I) = I: GOTO 55300
55260 IF PEEK (SL + 5) = 24 AND PEEK (SL + 7) = 56 THEN
SL$(I) = "Comm. card":SL(I) = I: GOTO 55300
55270 IF PEEK (SL + 5) = 56 AND PEEK (SL + 7) = 24 THEN
55274
55272 GOTO 55280
55274 IF TI(3) = 1 AND LEFT$(TI$(3),2) = "80" THEN SL$(I
) = TI$(3):SL(I) = I: GOTO 55300
55276 SL$(I) = "Serial card":SL(I) = I
55280 IF PEEK (SL + 11) = 1 THEN IF INT ( PEEK (SL + 12
) / 16) < 10 THEN SL$(I) = CD$( INT ( PEEK (SL + 12) /
16)):SL(I) = I: GOTO 55300
55290 IF PEEK (SL + 5) = 72 AND PEEK (SL + 7) = 72 THEN
SL$(I) = "Parallel card":SL(I) = I
55300 NEXT I
55302 REM * ID DRIVES
55305 FOR I = 48946 TO 48946 + PEEK (48945)
55310 DD = PEEK (I): IF DD > = 128 THEN DD = DD - 128
55315 SL = 7

```

```

55320 IF DD > = 16 * SL THEN DD = DD - (16 * SL): GOTO
      55335
55325 SL = SL - 1: IF SL > 0 THEN 55320
55330 GOTO 55350
55335 IF DD = 4 THEN SL$(SL) = "ProFile":SL(SL) = SL
55340 IF DD = 0 THEN SL$(SL) = "Disk drive":SL(SL) = SL
55350 NEXT I
55400 PRINT
55405 PRINT "SLOT #";: HTAB 9: PRINT "Description";: HTAB
      29: PRINT "Slot"
55407 PRINT "-----";: HTAB 9: PRINT "-----";: HTAB
      29: PRINT "----"
55410 FOR I = 1 TO 7
55420 PRINT "Slot ";I;":": HTAB 9: PRINT SL$(I): HTAB 29
      : PRINT SL(I)
55430 NEXT I
55450 VTAB 20: HTAB 11: GOSUB 560
59995 RETURN
60000 ER = PEEK (222): POKE 216,0:SW = 1
60010 VTAB 18: GOSUB 8970: VTAB 20: PRINT "Error <;ER;">
      occurred: ";
60020 ON ER GOTO 60100,60200,60300,60400,60500,60600,60100
      ,60800,60900,61000,61100,61200,61300,61400,61500,61600
      ,61700,61800,61900,62000,62100
60030 END
60100 PRINT "ERROR UNKNOWN."
60195 END
60200 PRINT "RANGE ERROR."
60295 END
60300 PRINT "NO DEVICE CONNECTED."
60395 END
60400 PRINT "WRITE PROTECTED."
60495 END
60500 PRINT "END OF DATA."
60510 VTAB 21: HTAB 11: GOSUB 560
60520 IF EP = 11 THEN CALL - 3288:TF = 1: PRINT D$;"CLOS
      E": PRINT D$;"DELETE ";F$: GOTO 490
60595 END
60600 PRINT "PATH NOT FOUND."
60610 VTAB 21: HTAB 11: GOSUB 560
60620 IF EP = 1 THEN CALL - 3288: GOTO 15060
60622 IF EP = 2 OR EP = 3 THEN CALL - 3288: GOTO 25060
60624 IF EP = 4 THEN CALL - 3288: GOTO 30060
60626 IF EP = 5 THEN CALL - 3288: GOTO 35060
60628 IF EP = 6 THEN CALL - 3288: GOTO 40060
60630 IF EP = 7 THEN CALL - 3288: GOTO 45060
60632 IF EP = 8 THEN CALL - 3288: GOTO 50060
60640 IF EP = 10 THEN CALL - 3288: GOTO 10000
60695 END
60700 PRINT "ERROR UNKNOWN."
60710 VTAB 21: HTAB 11: GOSUB 560
60720 CALL - 3288: GOTO 9000
60795 END

```

```

60800 PRINT "I/O ERROR."
60810 VTAB 21: HTAB 11: GOSUB 560
60820 CALL - 3288: GOTO 9000
60895 END
60900 PRINT "DISK FULL."
60910 VTAB 21: HTAB 11: GOSUB 560
60920 CALL - 3288: GOTO 9000
60995 END
61000 PRINT "FILE LOCKED."
61095 END
61100 PRINT "INVALID OPTION."
61195 END
61200 PRINT "NO BUFFERS AVAILABLE."
61295 END
61300 PRINT "FILE TYPE MISMATCH."
61395 END
61400 PRINT "PROGRAM TOO LARGE."
61495 END
61500 PRINT "NOT DIRECT COMMAND."
61595 END
61600 PRINT "SYNTAX ERROR."
61695 END
61700 PRINT "DIRECTORY FULL."
61795 END
61800 PRINT "FILE NOT OPEN."
61895 END
61900 PRINT "DUPLICATE FILE NAME."
61905 VTAB 21: PRINT "You WILL destroy old ";F$
61910 V = 22: GOSUB 580
61915 VTAB 22: HTAB 25: CALL - 958: GET Q$: PRINT Q$
61920 IF LEFT$(Q$,1) = "Y" OR LEFT$(Q$,1) = "y" THEN
61950
61925 IF LEFT$(Q$,1) = "N" OR LEFT$(Q$,1) = "n" THEN
61940
61930 GOSUB 520: GOTO 61915
61940 IF EP = 1 THEN CALL - 3288: GOTO 9000
61942 IF EP = 10 THEN CALL - 3288: GOTO 9000
61950 IF EP = 1 THEN CALL - 3288: GOTO 15360
61952 IF EP = 10 THEN CALL - 3288: GOTO 10630
61995 END
62000 PRINT "FILE BUSY."
62095 END
62100 PRINT "FILE(S) STILL OPEN."
62195 END

```

Figure 10.13. XMAS.PROG program.

The rest of this section contains explanations of the program code.

<i>Line</i>	<i>Description</i>
Line 1-9	Explained earlier.
Line 10	Explained earlier.
Line 15	Explained earlier.
Line 20	Dimension statements.
Line 30	Dimension statements.
Line 70	Mark all of the slot data as empty.
Line 80	Read all of the menu data.
Line 82-88	Data statements.
Line 90	Read system configuration default data.
Line 92	Default system data.
Line 94	True-False variable and Read-File variable set to zero.
Line 400-405	Read the current prefix.
Line 415-440	Read system file if it already exists.
Line 490	Activate the subroutine at line 55000.
Line 495	Make a hard branch to the main driver.

The following code descriptions are for the general and specific subroutines in the XMAS.PROG.

<i>Line</i>	<i>Description</i>
Line 500	General prompting subroutine.
Line 510	General error message subroutine.
Line 520	General delay loop subroutine.
Line 530	Print the date subroutine.
Line 540	Print the current prefix subroutine.
Line 550	General "correct entry?" prompt subroutine.
Line 560	General prompt to continue subroutine.
Line 570	Escape key capability subroutine.
Line 580	Delete prompt subroutine.
Line 590	General "any changes?" prompt subroutine.
Line 600	Another entry prompting subroutine.
Line 1395-1490	String filter subroutine.
Line 1495-1600	Number filter subroutine.
Line 1999-2140	Read records subroutine.
Line 2495-2640	Write records subroutine.
Line 2999-3070	Print screen subroutine.
Line 3100-3250	System configuration subroutine.
Line 5000-5295	Pathname validation subroutine.

Line 5300–5360 “Printer set up?” prompting subroutine.  
 Line 8899–8925 Report heading subroutine.  
 Line 8950–8960 Print screen title subroutine.  
 Line 8970 Print a line of dashes across screen subroutine.

The main driver code was explained in detail earlier.

The following code is for the purpose of specifying your system configuration.

<i>Line</i>	<i>Description</i>
Line 10000	Clear the screen.
	Print the top two lines of the screen.
Line 10010	Print the title line on the screen.
Line 10020	Complete printing the title area on the screen.
Line 10030	Print the date on the screen.
Line 10040	Print the current prefix on the screen.
Line 10050	Print the ESC key capability on the screen.
Line 10055	Test RF value for a value of one.
	If true go to line 10100.
Line 10060	Invoke the pathname subroutine.
Line 10070	Test the return from the pathname subroutine to determine if an ESC key was typed.
	Note: The code lines from 10000 through 10070 are the same code as is used for each major subroutine. The only difference is line xxx10 for each menu selection. You can see this with the lines of code that start at lines 15000, 20000, 25000, 30000, 35000, 40000, 45000, and 50000. Since the code is the same for all of these areas except one line each, none of these will be further explained.
Line 10100	Print the RETURN key for the acceptance of default values.
Line 10110	Position cursor and clear the screen.
Line 10120	Invoke the subroutine at line 3100.
Line 10200–10250	Present the default boot and data drive values on the screen and wait for operator response. Then test the response for a value different from the default.
Line 10300–10350	Present the default boot and data slot values on the screen and wait for operator response. Then test the response for a value different from the default.
Line 10400–10450	Present the default printer slot and type values on the screen and wait for operator response. Then test the response for a value different from the default.
Line 10500	Prompt operator for any changes to the selections just made.
Line 10510	Test for an ESC key character having been typed.
	If ESC key was typed go to line 14995 (end of subroutine).

Line 10520–10540	Test for legal Y or N answers. If responses are not valid, trap response, inform operator, and then give operator another chance.
Line 10600	Position cursor and inform operator that screen information is being saved.
Line 10610–10690	Set up error handler flag and save system default values as a sequential text file named SYSTEM to the diskette.
Line 10700	Set TF variable (True-False) to a zero.
Line 14995	Reset switch and return from subroutine.

The following section of code is for the purpose of creating a file.

<i>Line</i>	<i>Description</i>
Line 15000–15070	Set up the file and prompt for the file pathname.
Line 15300	Clear screen from line 9 down.
Line 15310	Set up error handler routine.
Line 15330	Define record pointer values.
Line 15340	Create the specified text file.
	Note: When you use the CREATE command there will be no record length stored on the diskette. Therefore, the create command was used to determine whether the file had been created before. This is similar to the VERIFY command except CREATE works with positive logic, whereas VERIFY requires negative logic for the determination of duplicate files.
Line 15360	Delete the file just created.
Line 15380	Write record zero for the file just specified.
Line 15420	Turn off error flag.
Line 15430	Test value of switch. Go to line 19995.
Line 15500	Set switch and go to line 9000.
Line 19995	Return from subroutine.

The following code is for the purpose of deleting an entire file.

<i>Line</i>	<i>Description</i>
Line 20000–20070	Set up the screen and prompt for the file pathname.
Line 20100	Clear screen from line 9 down.
Line 20110–20140	Ask operator if deleting the specified file is really what is wanted. Prompt operator to answer with a Y or N.
Line 20150–20180	Test for an ESC key having been typed. Test for a Y or N answer. Trap out all other answers.
Line 20200	Actually delete the file specified.
Line 24995	Return from subroutine.

The following code is for the purpose of adding a new record.

<i>Line</i>	<i>Description</i>
Line 25000–25070	Set up the screen and prompt for the file pathname.
Line 25300	Clear the screen from line 7 down.
Line 25305	Set entry point to 3. Set error flag.
Line 25310	Read record zero of specified file.
Line 25315	Clear the error flag.
Line 25350–25360	Determine length of the file name.
	Format the screen with file name data.
Line 25370	Clear the screen and present required file information on the screen.
Line 25380–25410	FOR—NEXT I loop that cycles through all required data for the specified file.
	Position cursor for each piece of file information.
	Assign entered data to the correct variable.
	On each cycle test for an ESC key typed.
Line 25415	Test switch, SW, for the ESC key typed.
Line 25420	Clear bottom of the screen.
	Ask operator if there are any changes to be made.
Line 25425–25450	Test response for an ESC key, Y key, or N key.
	Trap out all other keys on the keyboard.
Line 25500	Clear bottom of screen and ask operator which item is to be changed.
Line 25502–25510	Test the response.
	Validate for a correct answer.
Line 25520–25540	Position cursor.
	Capture entry made.
	Branch back and ask if there are any other changes.
Line 25600–25630	Position cursor and fill screen for the entry of card receptions.
Line 25700	Set index to zero.
Line 25710–25815	Response filter that accepts only a Y or N for data input.
Line 25815	Branch to line 25710.
Line 25840	Print a blank line to complete line of text and reposition cursor to the left margin on the screen.
Line 25850–25880	Position cursor and ask operator for any changes.
	Test for all valid answers and trap out invalid responses.
Line 25900–25915	Set up error handler.
	Write a record to the end of the file.
	Clear error flag.
Line 25920	Set up error handler.
Line 25925	Reset record zero pointer values.
	Write record zero values.
Line 25930	Turn off error flag.
Line 25950–25970	Position cursor.

Ask operator if there is another entry.  
 Trap all but valid answers.  
 Line 29995      Return from subroutine.

The next section of code is for the editing of a record.

<i>Line</i>	<i>Description</i>
Line 30000-30070	Set up screen and prompt for the file pathname.
Line 30300	Clear the screen from line 7 down.
Line 30305	Set entry point to 4. Set error flag.
Line 30310	Read record zero of specified file.
Line 30315	Clear the error flag.
Line 30350-30360	Present file information onto the video screen.
Line 30370-30380	Ask operator to enter the record to be edited.
Line 30382-30385	Test for the ESC key being pressed.
	Test and trap the range of the number entered.
Line 30390-30395	Reformat the screen with the new information.
Line 30400	Read the specified record.
Line 30410	Clear the screen from line 9.
Line 30420-30440	Format the screen with the information that needs to be edited.
	Ask the operator if there are changes to be made.
Line 30450-30480	Test answer for the ESC key for backup.
	Test for the Y or N answer.
Line 30500	Ask operator which item is to be changed.
Line 30502	Test for the ESC key backup.
Line 30505-30540	Enter the new data and test for its validity.
	Note: When entering or changing the years when cards are received, you simply position the cursor, type a Y or N, and then go on to the next item. Use the arrow keys to move the cursor. When you type a carriage return, the routine will be exited.
Line 30600-30815	Enter the years when Christmas cards were received.
Line 30840	Print a blank line to reset the cursor back to the left margin.
Line 30850	Prompt the operator for a response.
Line 30855-30880	Test and trap for a valid answer.
Line 30900	Clear the bottom of the screen.
Line 30910	Rewrite the record to the file.
Line 30950	Ask the operator if there is another entry.
Line 30955-30970	Test and trap the answer for a valid and correct answer.
Line 34995	Return from subroutine.



The next code is for the purpose of deleting a record.

Line 35000-35070	Set up the screen and prompt for the file pathname.
Line 35300	Clear the screen from line 7 down.
Line 35305	Set entry point to 5. Set error flag.
Line 35310	Read record zero of specified file.
Line 35315	Clear the error flag.
Line 35350-35395	Reformat screen and ask operator for the record to be deleted. Ask for the record to be deleted. Test the record number entered for a valid record. Finally, reformat the screen with the new information.
Line 35400	Read the specified record.
Line 35410	Clear the screen.
Line 35420-35470	Present record information on the screen.
Line 35480	Ask operator if record is to be deleted.
Line 35490-35510	Test to determine if ESC key was typed. Test for a Y or N answer. Trap out all other keys.
Line 35600	Set record fields to question marks to signify deletion.
Line 35610-35630	Reformat screen with deleted field data and print on the screen.
Line 35640	Inform operator that record is being deleted.
Line 35645	Rewrite specified record.
Line 35650	Ask operator if there is to be another record deleted.
Line 35660-35690	Test for an ESC key typed. Test for a Y or N answer. Trap out all other responses.
Line 39995	Return from subroutine.

<i>Line</i>	<i>Description</i>
Line 40000-40070	Set up the screen and prompt for the file pathname.
Line 40300	Clear the screen from line 7 down.
Line 40305	Set entry point to 6. Set error flag.
Line 40310	Read record zero of specified file.
Line 40315	Clear the error flag.
Line 40350-40380	Clear screen and print record data to be reviewed. Ask operator which record is to be reviewed.
Line 40385	Test request for a valid record number.
Line 40390-40395	Reformat the screen with the new information.
Line 40400	Read the specified record.
Line 40410	Clear the screen from line 9 down.

Line 40420	Present screen information on the screen.
Line 40430	Print the record information.
Line 40440	Clear the bottom portion of the screen.
Line 40450-40470	Format and print card reception information on the screen.
Line 40480	Ask if another record is to be reviewed.
Line 40490-40510	Test for ESC key being typed.
	Test for a Y or N answer.
	Trap out all other responses.
Line 44995	Return from subroutine.

<i>Line</i>	<i>Description</i>
-------------	--------------------

Line 45000-45070	Set up the screen and prompt for the file pathname.
Line 45100	Printer set-up subroutine.
Line 45160	Clear screen and ask operator to press any key.
Line 45170	Read record zero of specified file.
Line 45180	Print the number of records in the file.
Line 45200	Activate printer slot.
Line 45210	If printer type is parallel, set up printer buffer.
Line 45300	Assign page value to 1.
	Print report header to the printer.
Line 45310	Top of printing FOR—NEXT I loop.
Line 45315	Set up entry point.
	Set error flag.
Line 45320	Set record number value.
	Read record from the diskette.
Line 45325	Turn off error flag.
Line 45330-45420	Print report item to the printer.
Line 45430	Increment line count.
Line 45440	Test line count for 50.
Line 45450	Print blank lines to the top of next page.
Line 45460	Print report header.
Line 45480	Range of the I FOR—NEXT loop.
Line 45485	Print blank lines to the top of next page.
Line 45490	Reset output to the video screen.
Line 45495	Turn off printer slot.
Line 49995	Return from subroutine.

<i>Line</i>	<i>Description</i>
-------------	--------------------

Line 50000-50070	Set up the screen and prompt for the file pathname.
Line 50100	Printer set-up subroutine.
Line 50160	Prompt operator to press any key.

Line 50165	Turn off error flag.
Line 50170	Read record zero from diskette.
Line 50180	Inform operator of the number of records in the file.
Line 50200	Activate printer slot.
Line 50210	If printer type is parallel, set up printer buffer.
Line 50310	Top of the FOR—NEXT I printer loop.
Line 50320	Assign record number.
	Read record from diskette.
Line 50325–50370	Print mailing label.
Line 50400	Range of the I loop.
Line 50410	Reset output to the video screen.
Line 50420	Turn off printer slot.
Line 54995	Return from subroutine.

The following code informs you of your systems resources.

<i>Line</i>	<i>Description</i>
Line 53000–53014	Set up the screen and prompt for the file pathname.
Line 55015	Capture machine ID byte.
Line 55020–55050	Determine computer type.
Line 55060–55100	Determine memory size.
Line 55110–55160	Determine 80-column card and clock/calendar card.
Line 55185–55300	Determine slot occupancy.
Line 55302–55350	Identify the disk drives installed.
Line 55400–55430	Print the results on the video screen.
Line 55450	Prompt operator to press any key.
Line 59995	Return from subroutine.

In the next major area of code is the error handler routines. The driver routine for the error handler is located in lines 60000 through 60030. Line 60020 uses the ON—GOTO (CASE) construct to branch to the error condition that occurred. You will notice that many of the error routines have nothing in them except the printing of the error message. This was done on purpose to give you the opportunity to enter your own error handling.

You will notice that the code in lines 60600 through 60695 has been written to show you one way of handling errors. The variable EP (Entry Point) provides a very convenient way of determining the area where an error occurred. By setting the entry point to various values before setting the ONERR flag, you will have complete control of your program at all times.

You will also notice that extensive use of the CALL —3288 has been made. This is because it is recommended that the processor stack be cleared when you branch out of a subroutine to handle an error or other evolution. By using this call, you can prevent the embarrassment of losing control of your program.

<i>Line</i>	<i>Description</i>
Line 60000	Capture the error that has occurred. Clear the error flag. Set the switch.
Line 60010	Print the error number that has occurred.
Line 60020	Branch to the correct error handler based upon the error that occurred.

Only a few of the error handler routines are explained here.

<i>Line</i>	<i>Description</i>
Line 60500	Print error message.
Line 60510	Prompt operator to press a key to continue.
Line 60520	Test the entry point. Perform clean-up of stack. Set TF variable. Close all files. Delete specified file. Branch to top of program.
Line 60595	End the program.
Line 60600	Print error message.
Line 60610	Prompt operator to press a key to continue.
Line 60620	Is the entry point 1?
Line 60622	Is the entry point 2 or 3?
Line 60624	Is the entry point 4?
Line 60626	Is the entry point 5?
Line 60628	Is the entry point 6?
Line 60630	Is the entry point 7?
Line 60632	Is the entry point 8?
Line 60640	Is the entry point 10?
Line 60695	End the program.
Line 60700	Print error message.
Line 60710	Prompt operator to press any key to continue.
Line 60795	End the program.
Line 60800	Print error message.
Line 60810	Prompt operator to press any key to continue.
Line 60895	End the program.

Line 60900	Print error message.
Line 60910	Prompt operator that press any key to continue.
Line 60995	End the program.
Line 61900	Print error message.
Line 61905	Inform operator that the old file will be destroyed.
Line 61910	Ask if that is what is wanted.
Line 61915-60930	Trap the answer for correctness.
Line 61940	Is entry point 1?
Line 61942	Is entry point 10?
Line 61950	Is entry point 1?
Line 61952	Is entry point 10?
Line 61995	End the message.

This concludes the explanation of the XMAS.PROG.

```

63000 D$ = CHR$ (4)
63010 PRINT D$;"OPEN MAIL.PROG"
63020 PRINT D$;"WRITE MAIL.PROG"
63030 LIST
63040 PRINT D$;"CLOSE"
63050 END

```

Figure 10.14. EXEC file code.

The small segment of code shown in Figure 10.14 is the code necessary to turn XMAS.PROG into a text file. This was done for the purpose of allowing me to add, edit, and move code around easily through the use of a word processor. This way, the program could be written quickly using the word processor. Then the text file was EXECuted. This left me with a program that could be exercised and debugged, if necessary. Further, it allowed for easy re-numbering of code and adding additional code. You may want to try to write programs this way.

## SUMMARY

This chapter has been primarily devoted to the program XMAS.PROG. This program is actually a small management information system that provides you with a way of managing your Christmas card list.

Early in the chapter, you were given some of the principles of good program design and general rules for selecting a problem for computer solution. Even though very little time was

spent discussing what goes into good program design, the principles involved and the procedures to be followed are very sound, easy to follow, and provide help in developing programs regardless of their physical size.

A number of levels of correctness for a program were given. These were shown for the purpose of giving you a way of determining how well your programs may perform under varying conditions and operators.

A short discussion of menus with their advantages were given to show you why they are being used so often in programs.

The next section showed you the general logic flow for the organization of the XMAS.PROG. The interesting thing with this arrangement of code is that it applies to almost every program that you might develop.

The last two sections of this chapter were devoted to presenting code and code segments that were used to implement the XMAS.PROG program. Section 10.3 was nothing but the code for the program, along with explanations of what the code does.

## QUESTIONS

1. What are the requirements for selecting a problem for computer solution?
2. What are the levels of correctness and how may they be achieved?
3. Diagram the general logic flow of any program.
4. What are the advantages of using the arrangement of code as shown in this chapter?
5. What are the advantages of using menus to control a program?
6. Why was XMAS.PROG implemented using subroutines so heavily?
7. Looking at the XMAS.PROG, how could you improve the program?
8. If you improve the program, what price do you pay for the improvement in the way of code and operation?
9. The XMAS.PROG was written primarily using a word processor. Do you have any ideas on why this was done?

# APPENDIX A. DOS AND PRODOS COMPARISONS

*I distrust all systematisers,  
and avoid them. The will to a  
system shows lack of honesty.*

*Nietzsche, 1888*

## OVERVIEW

This appendix will summarize the differences between DOS 3.3 and ProDOS.

First of all, the various diskette formats are discussed briefly from the point of view of the different diskette formats you might have available. A procedure is given for determining the format of an unknown diskette.

The next section of this appendix discusses those DOS 3.3 commands that are no longer supported by ProDOS.

The last section of this appendix shows the ProDOS commands in ProDOS. Each command is shown with the options allowed, command examples, and a statement comparing the command to DOS 3.3

## THE DISKETTES

When a diskette is formatted using the DOS 3.3 command INIT, the programs and files on that diskette will use DOS 3.3 and can only use Apple Disk II drives.

When a diskette is formatted using the ProDOS FILER utility, the programs and files on that diskette will use ProDOS and can use all disk drives made by Apple Computer, Inc. for the Apple II family.

One of the more interesting problems that immediately arises is, "Which operating system is required by this disk?" There are a number of possibilities that exist. These are for the Apple II computers:

- ProDOS 1.0
- DOS 3.3
- DOS 3.2.1
- DOS 3.2
- PASCAL
- and for the Apple III computer
- SOS 1.1

If the diskette has a well-annotated diskette label, then your problem is probably solved. However, there are always those diskettes that you forgot to label or labeled incorrectly or inadequately.

So, let's go through a possible procedure to determine the formatting and operating system of the diskette.

Start by booting up your system with a ProDOS diskette installed in the boot drive that has the FILER program on the diskette. Then using the intelligent RUN command execute the FILER program. If you can read the contents of any diskette placed into a disk drive, it is a ProDOS diskette. Label this diskette so that you will not have to guess later.

If, however, you cannot read the diskette contents, then you will need to perform a warm start with DOS 3.3 installed in your boot drive. For example, PR#6. When you have booted DOS 3.3, try the CATALOG command to view the contents of the unknown diskette. If you can view the contents, you have a DOS 3.3 diskette that probably can be converted to ProDOS. Place a label on the diskette so it may be identified easily later. If this does not work, then you have one more option.

At this point, you will have to restart your system with the DOS 3.3 BASIC diskette installed into your boot drive. This diskette will tell you to install your DOS 3.2.1 diskette in order to complete the booting procedure. If your unknown diskette will finish the booting process, then you have a DOS 3.2 or DOS 3.2.1 diskette. This diskette will have to be



MUFFINed to DOS 3.3 and then converted to ProDOS. Place a label on the diskette stating it is a DOS 3.2 diskette.

If you are still unable to read the contents of the unknown diskette, then you probably have a PASCAL diskette, Apple III SOS 1.3, or a protected diskette that cannot be converted to ProDOS.

This gives you a simple, straightforward procedure for determining the recording format of a diskette.

## COMMANDS NO LONGER SUPPORTED

There are a number of DOS 3.3 commands that are no longer supported. These are:

- FP
- INT
- INIT
- MAXFILES
- MON
- NOMON

Since these commands are no longer valid, they will not be discussed.

There is one command that you may still use, but is really not supported. That command is the VERIFY command. You may still use this command without causing an error to occur.

## PRODOS COMMANDS SUPPORTED

In this section, you will be given the general form for the description of each command that is described in this book. Normally, the general form of a command is known as the syntax required for the command. A command's syntax for ProDOS looks generally like the following:

command [pn] [,S#] [,D#]

In general, you enter the command, followed normally with a pathname and then possibly with a series of options.

In all of the commands summarized below, this general form will be followed. For commands that are different for different file types, both syntactical forms are given.

HELP

NOHELP

HELP	e.g.,	HELP	Immediate mode only
		HELP PREFIX	
NOHELP	e.g.,	NOHELP	Immediate mode only

The HELP command gives you an on-line way of getting help with ProDOS commands, provided you have the HELP file and HELPSCREENS file installed on a currently active diskette.

Once the HELP command has been activated, you may deactivate that capability with the NOHELP command.

]— pn [,S#] [,D#]

—	e.g.,	— DUMMY	Immediate mode only
		— /PRODOS/DUMMY	

This new command is very handy; it is the intelligent RUN command. By using this command, you no longer need to know the type of program file you are executing. This command will determine the file type and automatically perform the proper RUN or BRUN command.

]RUN pn [,@#] [,S#] [,D#]

]— pn [,S#] [,D#]

RUN	e.g.,	RUN /PRODOS/DUMMY	Immediate and deferred
		RUN	
		— /PRODOS/DUMMY	

The ProDOS version RUN has only one added capability, which is that you may specify the line number at which execution is to take place. Notice that you may alternatively use the intelligent RUN command.

This command operates the same as in DOS 3.3 except for the added option.

]LOAD pn [,S#] [,D#]

LOAD	e.g.,	LOAD /PRODOS/VIEW	Immediate and deferred
------	-------	-------------------	------------------------

This command allows you to LOAD an Applesoft II BASIC program file into memory.

**This command operates the same as in DOS 3.3.**

**]SAVE pn [,S#] [,D#]**

<b>SAVE</b>	<b>e.g.,</b>	<b>SAVE /MY.DISK/DEMO</b>	<b>Immediate and deferred</b>
-------------	--------------	---------------------------	-------------------------------

**This command allows you to SAVE an Applesoft II BASIC program to a diskette file. This command operates the same as in DOS 3.3.**

]CAT [pn] [ ,S#] [ ,D#]

CAT	e.g.,	CAT /PRODOS CAT /PRODOS,S6,D1	Immediate and deferred
-----	-------	----------------------------------	------------------------

This command will present an abbreviated set of information to your video screen in a 40-column format. This command and the CATALOG command will display the contents of a single directory. It is necessary that you specify the name of the directory whose contents you want displayed.

**If you do not specify a directory name, then the prefix directory will be displayed.**

]CATALOG [pn] [,S#] [,D#]

CATALOG	e.g.,	CATALOG /PRODOS	Immediate and deferred
		CATALOG /PRODOS,S6,D1	

If you have an 80-column format video screen selected, then this command will display the maximum normal information available through ProDOS.

This command will display the logical end of a file, the load address of a binary file, or the file's record length in the case of a random-access file.

**This command operates the same as in DOS 3.3 except for the added capabilities.**

**]PREFIX [pn] [,S#] [,D#]**

<b>PREFIX</b>	e.g.,	<b>PREFIX /PRODOS/STATES</b>	Immediate and deferred
<b>PREFIX</b>	e.g.,	<b>PREFIX /PRODOS</b>	Immediate and deferred
		<b>PREFIX /S6,D1</b>	

This command allows you to set the name of the directory that contains the files that you wish to access. Once you have set the prefix, all files accessed must be stored in that directory.

If the prefix variable is empty, the files accessed will be those located in the main directory of the last referenced drive and slot. This is exactly the same as with DOS 3.3.

]CREATE pn [,Ttype] [,S#] [,D#]

CREATE	e.g.,	CREATE PIC4,TBIN CREATE DIRECTORY,TDIR	Immediate and deferred
--------	-------	---	------------------------

This command is for the purpose of creating files. However, it will be used primarily for the purpose of creating directories. The BAS, TXT, BIN type file are automatically created when you use the SAVE, OPEN, and BSAVE commands, respectively. The VAR file type is created when using the STORE command. Text files may also be created by using the APPEND command.

]RENAME pn1,pn2 [,S#] [,D#]

RENAME	e.g.,	RENAME STOCKS,PORTFOLIO	Immediate and deferred
--------	-------	-------------------------	------------------------

This command allows you to change the name of a file stored on a diskette.  
This command operates the same as in DOS 3.3.

]DELETE pn [,S#] [,D#]

DELETE	e.g.,	DELETE /MY/LOSSES DELETE /MY/DEBTS,S6,D2	Immediate and deferred
--------	-------	---	------------------------

This command allows you to DELETE any file stored on a diskette.  
This command operates the same as in DOS 3.3.

]LOCK pn [,S#] [,D#]

]UNLOCK pn [,S#] [,D#]

LOCK	e.g.,	LOCK /MY/PORTFOLIO	Immediate and deferred
UNLOCK	e.g.,	UNLOCK /MY/PORTFOLIO	Immediate and deferred

These commands allow you to change the protection status of files stored on a diskette. These commands operate the same as in DOS 3.3.

**]CHAIN pn [,@#] [,S#] [,D#]**

CHAIN	e.g.,	CHAIN PART.TWO	Immediate and deferred
-------	-------	----------------	------------------------

**You are now able to CHAIN one program to any line number of another program.**

This command operates the same as in DOS 3.3, except that now it works with Applesoft II BASIC.

**]STORE pn [,S#] [,D#]**

<b>STORE</b>	<b>e.g.,</b>	<b>STORE /MY/DEBTS</b>	<b>Immediate and deferred</b>
--------------	--------------	------------------------	-------------------------------

The **STORE** command is used to store all of the current program's variables into a file on a diskette.

**This command is not supported in DOS 3.3.**

**]RESTORE pn [,S#] [,D#]**

<b>RESTORE</b>	<b>e.g.,</b>	<b>RESTORE /MY/DEBTS</b>	<b>Immediate and deferred</b>
----------------	--------------	--------------------------	-------------------------------

**This command will add the contents of the STOREd file to the variables of the current program.**

**This command is not supported in DOS 3.3.**

JPR# slot

]PR# slot [,A#] |A#

**PR#** e.g., **PR#6** Immediate and deferred mode

PR#	e.g.,	PR#0 PR#1 PR#0	Immediate and deferred mode
-----	-------	----------------------	-----------------------------

This command allows you to control the slot to which characters are to be transferred and/or stored.

This command operates the same as in DOS 3.3.

]IN# slot

]IN# slot [,A#] |A#

IN#	e.g.,	IN#2	Immediate and deferred mode
IN#	e.g.,	IN#3	Immediate and deferred mode
		IN#0	

This command allows you to control the slot from which characters are to be received.  
This command operates the same as in DOS 3.3.

]OPEN pn [,S#] [,D#]

]OPEN pn [,L#] [,S#] [,D#]

OPEN	e.g.,	OPEN /MY/XMAS.LIST	Deferred mode only
		OPEN /MY/XMAS.LIST,S6,D2	
OPEN	e.g.,	OPEN EXAMPLES,L28	
		OPEN /MY/LIST, L200,S6,D1	

The OPEN command may now be used to open any type of file for access.

File buffers are now allocated and deallocated dynamically when a file is opened and closed, respectively.

This command operates the same as in DOS 3.3.

]READ pn [,F#] [,B#]

]READ pn [,R#] [,F#] [,B#]

READ	e.g.,	READ /MY/XMAS.LIST	Deferred mode only
	e.g.,	READ EXAMPLES,F4	
	e.g.,	READ /MY/ADDRESS.LIST,R5	
	e.g.,	READ LEDGER,R22,F3	

In DOS 3.3 the READ command only allowed you to use the B# option. In the ProDOS version you are allowed to use both the F# and B# options. So you may now pass over a specified number of fields plus the specified number of bytes BEFORE the READ actually takes place. This essentially makes the POSITION command obsolete when reading files.

This command operates the same as in DOS 3.3, except for the added capability.

]WRITE pn [,F#] [,B#]

]WRITE pn [,R#] [,F#] [,B#]

WRITE	e.g.,	WRITE /MY/XMAS.LIST	Deferred mode only
	e.g.,	WRITE /MY/XMAS.LIST,F3	
	e.g.,	WRITE EXAMPLES	
	e.g.,	WRITE /MY/LIST,S6,D1	

In DOS 3.3 the WRITE command only allowed you to use the B# option. In the ProDOS version you are allowed to use both the F# and B# options. So you may now pass over a specified number of fields plus the specified number of bytes BEFORE the WRITE actually takes place. This essentially makes the POSITION command obsolete when writing files.

This command operates the same as in DOS 3.3, except for the added capability.

]CLOSE [pn]

CLOSE	e.g.,	CLOSE	Immediate and deferred
		CLOSE /MY/XMAS.LIST	

The CLOSE command still closes files; however, it is more important that a CLOSE command be executed before leaving a program because of the possibility that data may be lost.

This command operates the same as in DOS 3.3.

]APPEND pn [,S#] [,D#]

]APPEND pn [,L#] [,S#] [,D#]

<b>APPEND</b>	e.g.,	APPEND /MY/XMAS.LIST APPEND /MY/LIST/L200,D1 APPEND EXAMPLES,L28	Deferred mode only
---------------	-------	--	--------------------

The APPEND command has two new capabilities. You can use this command to add data to the end of any type of file. Further, APPEND may be used to add data starting at the beginning of the record immediately following the last logical record in a random-access text file.

This command operates the same as in DOS 3.3, except for the added options.

]FLUSH [pn]

<b>FLUSH</b>	e.g.,	FLUSH /MY/XMAS.LIST FLUSH MY/EXAMPLES	Immediate and deferred
--------------	-------	--	------------------------

The use of this command allows you to completely empty the file buffer in memory. This assures that you have written all data to a file. If you use the command frequently, your program execution will slow down; however, you will be assured that all data has been saved to the diskette media.

]POSITION pn,F#

<b>POSITION</b>	e.g.,	POSITION /MY/XMAS.LIST,F3	Deferred mode only
-----------------	-------	---------------------------	--------------------

The READ and WRITE commands under ProDOS allow you to specify the number of fields and bytes to be read and skipped over or discarded. Therefore, the POSITION command is not required to be used by ProDOS. It was retained to provide compatibility with DOS 3.3.

This command works the same as in DOS 3.3.

]BLOAD pn [,A#] [,B#] [,L#|E#] [Ttype] [,S#] [,D#]



**BLOAD**      e.g.,      **BLOAD PIC1**      Immediate and deferred mode  
                               **BLOAD PIC1,A\$4000**  
                               **BLOAD PIC1,A\$4000,L\$2000**  
                               **BLOAD PIC1,A16384,E\$24575**

The **BLOAD** command has three new capabilities. This command may now be used to load the binary image of any type of file, not just binary files. You may now load any portion of a file, any specified number of bytes expressed either as a length option or beginning and ending address, into memory.

This command operates the same as in DOS 3.3, except for the added options.

<b>]BSAVE pn ,A# ,L# ,E# [,B#] [,Ttype] [,S#] [,D#]</b>
---

**BSAVE**      e.g.,      **BSAVE PIC1,A\$2000**      Immediate and deferred mode  
                               **BSAVE PIC1,A8192**  
                               **BSAVE PIC1,A\$2000,L\$2000**  
                               **BSAVE PIC1,A8192,E\$4000**

This command stores the contents of memory specified. The number of bytes that are to be saved may be specified either by specifying a starting address and ending address or by specifying a starting address and the number of bytes to be transferred.

This command operates the same as in DOS 3.3, except for the added options.

<b>]BRUN pn [,A#] [,B#] [,L# ,E#] [,S#] [,D#]</b>
---

<b>] – pn [,S#] [,D#]</b>
---------------------------

**BRUN**      e.g.,      **BRUN TONE.BEEP**      Immediate and deferred mode  
                               **BRUN BEEPER,A\$300**  
                               **– TONE.BEEP**

There are new capabilities added to this command. You now are able to load into memory any portion of a binary file and run that code. The number of bytes may be specified either by specifying the starting address and the number of bytes or the starting address and ending address. As an alternative, you may use the intelligent **RUN** command.

This command operates the same as in DOS 3.3, except for the added options.

<b>]EXEC pn [,F#] [,S#] [,D#]</b>
-----------------------------------

**EXEC**                    e.g.,                    **EXEC DUMMY**                    Immediate and deferred mode

This command is very useful in a number of instances. The possible uses were discussed in Chapter 7.➤

This command works the same as it does in DOS 3.3.

# APPENDIX B.

## PRODOS 1.0

### MEMORY MAP

*Memory is the diary that we  
all carry about with us.*

*Oscar Wilde, 1895*

#### **OVERVIEW**

This appendix shows the memory map for ProDOS 1.0.

Further, there is a discussion and explanation of page zero memory locations, along with a general explanation of the main memory usage on a page basis.

There is also a discussion of the high memory considerations when using your own machine-language routines.

Finally, a memory map for an Apple IIe is shown.

## HIGH MEMORY CONSIDERATIONS

When ProDOS boots, a 1K (\$0400) buffer is set aside for a temporary file buffer and then sets HIMEM. This buffer starts at the highest 1K memory boundary. Once this buffer has been established, the HIMEM value is set. This initial file buffer is used for those commands that perform an OPEN and CLOSE that is transparent to the operator. The CATALOG and CAT commands are typical examples. See Figure B.2.

Typically, HIMEM is set to 38912 = \$9600 for an Apple II Plus with 64K of memory.

When an Applesoft II BASIC program is running, the HIMEM setting is changed each time a new file is OPENed. ProDOS will lower HIMEM by 1K (\$0400). When the file is CLOSED, ProDOS releases that file's buffer memory and then raises the HIMEM setting.

*If machine-language programs are to work with ProDOS, you must move HIMEM only in 256 (\$0100) byte increments. You should do this only when there are no files open and no string variables declared. This means you do it early in a program.*

If you want machine-language routines to be memory resident with ProDOS and an Applesoft II BASIC program, these routines should be added to the BASIC system program.

*The only safe area in memory is the \$0300 through \$03CF. There are no other safe memory areas.*

In general, before a program is run, HIMEM is set to the highest value that the system bit map will allow. All unmarked blocks in the system bit map are available to Applesoft II BASIC. Assume now that you want a machine-language routine to remain in memory for more than a single program execution. You must have your program mark the appropriate 256-byte pages used in the system bit map. Machine-language routines that are protected in this way may remain in memory and active until you reboot your system.

## ZERO PAGE

This section will discuss some of the useful places in the zero page of memory that you may use to great effectiveness.

However, before getting into the individual memory locations, it seems only logical to talk briefly about the general organization of memory in the Apple II Plus or Apple IIe on a page basis. A page of memory is a grouping of 256, \$00-\$FF, memory locations.

**Page 0 (\$0000-\$00FF):** For the 6502 microprocessor, page 0 is very important. Stored on this page are those values, hooks, parameters, etc, that are to be used frequently by other software, such as Applesoft II BASIC, DOS 3.3, Integer BASIC, miniassembler, etc. This is because the 6502 has, as one of its addressing modes, one that uses page 0. By using page 0, a great speed advantage may be realized.

**Page 1 (\$0100-\$01FF):** This page is used primarily by the processor for its stack. This area is used by the 6502 for subroutine returns, interrupts, reentrant code, and temporary parameters. This area of memory should only be used by a very experienced programmer.

**Page 2 (\$0200-\$02FF):** Page 2 is used by the keyboard and for a general input buffer. Characters inputted from the keyboard are normally stored in page 2. It is from this area that they are made available to other routines.

**Page 3 (\$0300-\$03FF):** This is the first page of memory that may be used by programmers without fear of interfering with other absolutely needed machine code. Now that I have said that, I am going to take away the top portion of this page of memory. That is because the area from \$03D0 to \$03FF is used for jump commands and linkage vectors to other parts of memory and programs.

The addresses 1008 = \$03F0 through 1023 = \$03FF, 16 bytes, are used by the monitor and AUTOSTART ROM to provide vectors to other monitor routines and storage locations for user-defined functions. The memory locations, \$03D0-\$03EF, 32 bytes, are used by DOS 3.3.

The rest of page 3 is available to the user for machine language instructions of his own choosing.

The top 16 memory locations are really quite useful. Each of these are defined as follows:

<i>Decimal</i>	<i>Hexadecimal</i>	<i>Meaning</i>
1008-1009	\$03F0-\$03F1	Used by the AUTOSTART ROM monitor as the interrupt, BRK, instruction vector, address.
1010-1011	\$03F2-\$03F3	These addresses are for the RESET (soft entry) vector used by the AUTOSTART monitor.
1012	\$03F4	Power up indicator. If the logical exclusive or of 165 = \$A5 with the contents of 1011 = \$03F3 is equal to the contents of 1012 = \$03F4, the RESET (soft entry) vector is considered valid. If this condition does not exist, the monitor will go through a complete powerup cycle, as if it were a cold start.
1013-1015	\$03F5-\$03F7	Reserved for Applesoft II BASIC ampersand (&) instruction.
1016-1018	\$03F8-\$03FA	Reserved for the CTRL-Y instruction.
1019-1021	\$03FB-\$03FD	The storage location for the nonmaskable interrupt vector instruction.
1022-1023	\$03FE-\$03FF	The storage for the IRQ, interrupt vector.

You can take advantage of these locations to help control a program, especially the typing of the RESET key. If you want the computer to boot, as if the machine were just started every time the RESET key were typed, you could embed into a program the following line of code:

```
0 POKE 1011,224
```

If, however, you only want the computer to RUN the program over from the beginning again when RESET is typed, you can use the following code:

```
12 POKE 1010,102 : POKE 1011,213 : CALL -1169
```

This line of code will trap the typing of the RESET key and allow you to run your program over again automatically. CALL -1169 is explained later. You should also provide for setting your computer back to its normal condition. This can be accomplished with:

43255 POKE 1010,191 : POKE 1011,157 : CALL -1169

Your system is now back to its normal condition.

Pages 4-7 (\$0400-\$07FF): In these four pages there are 1024 memory locations. 960 of these are used for text and/or low resolution graphics. A normal screen uses a  $40 \times 24$  byte area. This corresponds to 960 spaces. This area contains eight triple lines. Each triple line is 128 bytes long.

To make things worse, each 1/3 of the triple line is in one of the thirds of the screen. For example, the first line is in the upper third of the screen. The second third of the triple-line is in the second third of the screen. The last third of the triple line is in the last third of the screen. When you are in the text mode, each character is made up of a 7-bit wide by 8-bit high area on the screen. Each character has one embedded blank column.

Pages 8-11 (\$0800-\$0BFF): This is the start of memory for an Applesoft II BASIC program and can be used as the secondary text page. However, it is rare that this is done, because each character MUST be poked into the proper memory location.

Pages 32-63 (\$2000-\$3FFF): This is the beginning of the high-resolution page 1. This area maps bits in essentially the same manner that the low-resolution screen maps bytes; with a triple line.

Pages 64-95 (\$4000-\$5FFF): This area is the high-resolution graphics page 2. Once again the layout of this page is the same as described before.

Pages 150-191 (\$9600-\$BFFF): This area is used for the Disk Operating System on a 48K Apple II Plus or Apple IIe.

Pages 192-207 (\$C000-\$CFFF): This memory area is used for the hardware I/O area.

Pages 208-255 (\$D000-\$FFFF): These are used by the Apple II Plus or Apple IIe monitor and Applesoft II BASIC interpreter.

The following group of useful memory locations is that located on what is known as page 0 of memory. Page 0 of memory contains those locations with numbers from 0 to 255 in decimal or \$0000 to \$00FF in hexadecimal. These locations have very special meaning for the 6502 microprocessor, because it can address them directly with a two-byte machine instruction.

<i>Decimal</i>	<i>Hex</i>	<i>Meaning</i>
0- 2	\$00-\$02	Applesoft II BASIC soft entry. OG is equivalent to CTRL-C. (A & D)
3- 5	\$03-\$05	Jump to \$F128. (A)
6- 9	\$06-\$09	Undefined memory space.
10- 12	\$0A-\$0C	Address of USR jump instruction. (A)
13- 19	\$0B-\$13	Applesoft II BASIC. (A)

20	\$14	Subscript flag. (A) \$00 = subscripts allowed \$80 = subscripts not allowed
21	\$15	General purpose counter or flag. (A)
22	\$16	Comparison type flag byte. (A) 1 = > ; 2 = = ; 3 = > = ; 4 = < ; 5 = < > ; 6 = < = Used by the routine at \$DF6A in the interpreter.
23- 24	\$17-\$18	Applesoft II BASIC general usage. (A)
25- 27	\$19-\$1B	Undefined memory space.
28	\$1C	HCOLOR byte. High-resolution color mask. (A)
31	\$1F	Monitor usage. (M)
32	\$20	Left edge text window. Range 0-39. (A & M)
33	\$21	Width of text window. Range 1-40. (A & M)
34	\$22	Top edge of text window. Range 0-22. (A & M)
35	\$23	Bottom of text window. Range 1-24 (A & M)
36	\$24	Horizontal cursor position. (A, M, and P)
37	\$25	Vertical cursor position. (A & M)
38	\$26	Used by ProDOS. Sector read buffer address. (M)
39	\$27	Scratch space. (RWTS in P & M)
40	\$28	Monitor pointer to current screen line. (M)
40- 41	\$28-\$29	BASL/BASH. (P & M)
42	\$2A	Segment merge counter. (P & M) Scratch space. (RWTS in P & M)
43	\$2B	Boot slot × 16. Only after initial boot. (M) Scratch space. (RWTS in P & M)
44	\$2C	End point of last HLIN, VLIN, or PLOT. (A, M, and I) Checksum from sector head. (P & M)
45	\$2D	End point of last VLIN or PLOT. (A, M, and I) Sector number. (P & M)
46	\$2E	Track number from sector header. (P & M)
47	\$2F	Volume number from sector header. (P & M)
48	\$30	Low-resolution COLOR value × 17. (A, M, and I)
49	\$31	Screen mode. (M)
50	\$32	Text output format. (M) 63 = \$3F -INVERSE      255 = \$FF -NORMAL 127 = \$7F -FLASH (243 = \$F3 must be set to 64 = \$40)
51	\$33	Prompt character. (A, I, M, and P) > = Integer      ] = Applesoft * = monitor      ! = mini-assembler
53	\$35	Drive number in high bit. (RWTS in P & M)
54	\$36	CSW address - low byte. (M)
55	\$37	CSW address - high byte. (M)
56	\$38	KSW address - low byte. (M)

57	\$39	KSW address - high byte. (M)
58- 59	\$3A-\$3B	Program counter save and control area. (A, D, & M)
60	\$3C	Workbyte. (RWTS in P, D, & M)
61	\$3D	Sector number. (ROM) (P, D, & M)
60- 61	\$3C-\$3D	Old memory move start location. (M, D, & P)
62- 63	\$3E-\$3F	Old memory move end location. (M, D, & P)
		Address of ROM sector read subroutine. (P, D, & M)
64- 65	\$40-\$41	New memory move start location. (M, D, & P)
		ProDOS image address. (on BOOT P)
66- 67	\$42-\$43	New memory move end location. (M, D, & P)
		Buffer address. (P & M)
68- 69	\$44-\$45	Numeric operand. (P, D, & M)
68	\$44	ProDOS usage. (M, D, & P)
69	\$45	Save accumulator. (M, D, & P)
70	\$46	Save X register. (M, D, & P)
71	\$47	Save Y register. (M, D, & P)
70- 71	\$46-\$47	Scratch space. (P & M)
72	\$48	Save P, processor status register. (M & P)
73	\$49	Save S, processor stack. (M & P)
72- 73	\$48-\$49	IOB address. (P)
74	\$4A	ProDOS usage. (P)
74- 75	\$4A-\$4B	LOMEN address for Integer BASIC. (I & P)
76- 77	\$4C-\$4D	HIMEN address for Integer BASIC. (I & P)
78	\$4E	Random number location. (M & P)
78- 79	\$4E-\$4F	Random number storage field. (A & M)
80	\$50	High-resolution graphics shape temporary. (A & M)

80-249 \$50-\$F9 -Applesoft II BASIC storage locations. (A)
---

80- 81	\$50-\$51	High-resolution delta X for HLIN shape. (A & M)
82-102	\$52-\$66	Applesoft II BASIC and monitor. (A & M)
103-104	\$67-\$68	Start of program for Applesoft II BASIC. (A) Usually 2049 = \$0801
105-106	\$69-\$6A	Start of variable table address space. (A)
107-108	\$6B-\$6C	Start of array address space. (A)
109-110	\$6D-\$6E	End of variable table address space. (A)
111-112	\$6F-\$70	Start of free string storage space. (A)
113-114	\$71-\$72	General pointer. (A)
115-116	\$73-\$74	HIMEM address location. (A)
117-118	\$75-\$76	Line number being executed. (A)
119-120	\$77-\$78	Line where program terminated. (A)



121-122	\$79-\$7A	Line currently being executed. (A)
123-124	\$7B-\$7C	Line number where DATA is being READ. (A)
125-126	\$7D-\$7E	Current DATA address location. (A)
127-128	\$7F-\$80	Current INPUT or DATA address. (A)
129-130	\$81-\$82	Last used variable name. (A)
131-132	\$83-\$84	Last used variable address. (A)
133-171	\$85-\$AB	Applesoft II BASIC. (A)
172-174	\$AC-\$AE	General use flags/pointers. (A)
175-176	\$AF-\$B0	Address of end program. (A)
177-183	\$B1-\$B7	Applesoft II BASIC. (A)
184-185	\$B8-\$B9	Pointer to last character obtained through the character input routine. (A)
202-203	\$CA-\$CB	Start of INTEGER BASIC program. (I & A)
204-205	\$CC-\$CD	End of INTEGER variable storage. (I & A)
214	\$D6	RUN flag. (A) if > 127 = AUTO RUN.
215	\$D7	Undefined memory space.
216	\$D8	ONERR flag. (A) 0 = not set.
217	\$D9	ProDOS usage for the direct-deferred mode usage. (P)
216-217	\$D8-\$D9	INTEGER line number. (I & A) ONERR flag (A)
218-219	\$DA-\$DB	Line where ONERR occurred. (A)
220-221	\$DC-\$DD	Text pointer for error handler subroutine. (A)
222	\$DE	ONERR error code storage. (A)
223	\$DF	Stack pointer value before an error occurs. (A)
224-225	\$E0-\$E1	X coordinate of last HPLOT. (A)
226	\$E2	Y coordinate of last HPLOT. (A) (See 224-225 for X coordinate. (A) )
227	\$E3	Undefined memory space. (A)
228	\$E4	HCOLOR value. (A) 0 = 0      1 = 42      2 = 85      3 = 127 4 = 128    5 = 170    6 = 213    7 = 255
229	\$E5	General purpose high resolution usage. (A)
230	\$E6	High resolution plotting page. (A) 1 = 32    2 = 64    3 = 96
231	\$E7	SCALE value in high resolution. (A)
232-233	\$E8-\$E9	Start of shape table address. (A)
234	\$F1	High resolution collision counter. (A)
235-239	\$EB-\$EF	Undefined memory space.
241	\$F1	SPEED value. 256 - SPEED. (A)
242	\$F2	Applesoft II BASIC. (A)
243	\$F3	FLASH mask. (A)

		64 = FLASH (with 50 set to 127)
244-247	\$F4-\$F7	Applesoft II BASIC. (A)
248	\$F8	Stack pointer saved here before each statement. (A)
249	\$F9	ROT value in high-resolution graphics. (A)
250-255	\$FA-\$FF	Undefined memory space.

Legend: (A) = Applesoft II BASIC  
 (D) = Disk Drivers  
 (P) = ProDOS  
 (M) = Monitor  
 (I) = Integer BASIC

Figure B.1. Zero page usage.

Thus far a number of page 0 locations and their usage have been shown. I don't claim that you have all of them. In fact there are a number of blank spaces shown.

You can always find the value, in decimal notation, stored in any two-byte address location by executing the following code:

```
JPRINT PEEK(low byte) + PEEK(high byte) * 256
```

Since this is in the immediate mode, the calculated value will be presented to the screen in decimal form.

If you need to use zero-page memory locations for your routines, try to pick unused locations. However, if that is not possible, then pick an area on page zero, save the contents somewhere, load your routine or values, execute it, and finally restore the original contents.

## GENERAL MEMORY MAP

How memory is arranged when ProDOS is booted at startup time is shown for the Apple IIe with 128K of memory in the following table. The way your system memory is configured depends upon the physical memory available. If you have a 48K Apple computer with a 16K language card installed, memory allocation is different than it would be if you have an Apple IIe with 128K of total memory. This is shown in Figure B.2.

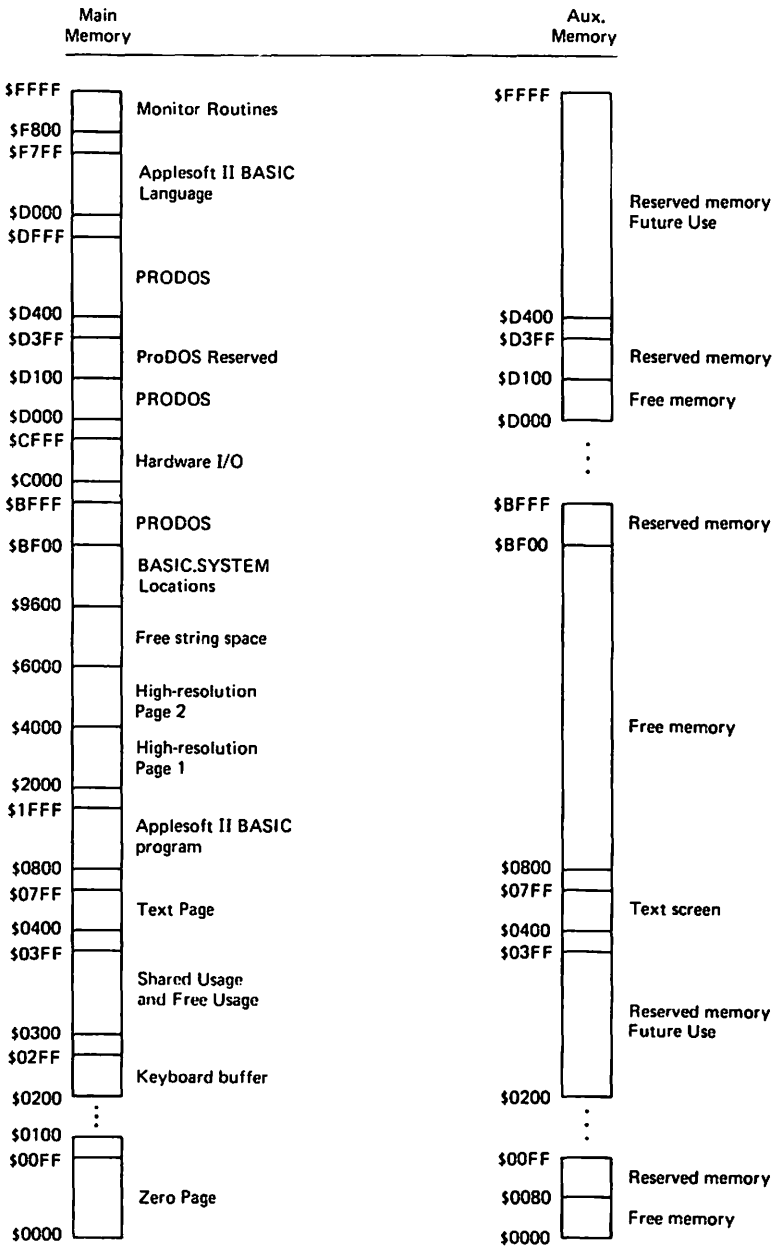
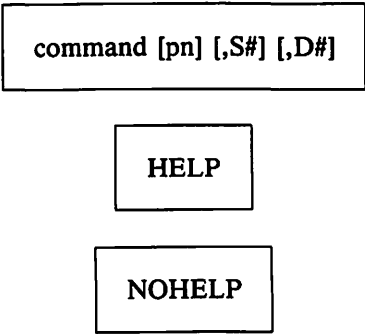


Figure B.2. Apple IIe memory map.

# APPENDIX C. PRODOS COMMAND SUMMARY

*New faces have more authority  
than accustomed ones.*

*Euripides, 426 B.C.*



HELP	e.g.,	HELP	Immediate mode only
NOHELP	e.g.,	HELP PREFIX	
		NOHELP	Immediate mode only

]— pn [,S#] [,D#]

—	e.g.,     — DUMMY — /PRODOS/DUMMY	Immediate mode only
---	--------------------------------------	---------------------

]RUN pn [,@#] [,S#] [,D#]

]— pn [,S#] [,D#]

RUN	e.g.,     RUN /PRODOS/DUMMY RUN — /PRODOS/DUMMY	Immediate and deferred Applesoft II BASIC
-----	---	--

]LOAD pn [,S#] [,D#]

LOAD	e.g.,     LOAD /PRODOS/VIEW	Immediate and deferred
------	-----------------------------	------------------------

]SAVE pn [,S#] [,D#]

]CAT [pn] [,S#] [,D#]

CAT	e.g.,     CAT /PRODOS CAT /PRODOS,S6,D1	Immediate and deferred
-----	--	------------------------

]CATALOG [pn] [,S#] [,D#]

CATALOG	e.g.,     CATALOG /PRODOS CATALOG /PRODOS,S6,D1	Immediate and deferred
---------	--	------------------------

]PREFIX [pn] [,S#] [,D#]

PREFIX	e.g.,     PREFIX /PRODOS/STATES	Immediate and deferred
--------	---------------------------------	------------------------

PREFIX	e.g.,	PREFIX /PRODOS PREFIX /S6,D1	Immediate and deferred
		<div>]CREATE pn [,Ttype] [,S#] [,D#]</div>	
CREATE	e.g.,	CREATE PIC4,TBIN CREATE DIRECTORY,TDIR	Immediate and deferred
		<div>]RENAME pn1,pn2 [,S#] [,D#]</div>	
RENAME	e.g.,	RENAME STOCKS,PORTFOLIO	Immediate and deferred
		<div>]DELETE pn [,S#] [,D#]</div>	
DELETE	e.g.,	DELETE /MY/LOSSES DELETE /MY/DEBTS,S6,D2	Immediate and deferred
		<div>]LOCK pn [,S#] [,D#]</div>	
		<div>]UNLOCK pn [,S#] [,D#]</div>	
LOCK UNLOCK	e.g., e.g.,	LOCK /MY/PORTFOLIO UNLOCK /MY/PORTFOLIO	Immediate and deferred Immediate and deferred
		<div>]CHAIN pn [,@#] [,S#] [,D#]</div>	
CHAIN	e.g.,	CHAIN PART.TWO	Immediate and deferred
		<div>]STORE pn [,S#] [,D#]</div>	
STORE	e.g.,	STORE /MY/DEBTS	Immediate and deferred

**]RESTORE pn [,S#] [,D#]**

**RESTORE**      e.g.,      **RESTORE /MY/DEBTS**      Immediate and deferred

**]PR# slot**

**]PR# slot [,A#]|A#**

**PR#**      e.g.,      **PR#6**      Immediate and deferred mode

**PR#**      e.g.,      **PR#0**  
**PR#1**      Immediate and deferred mode  
**PR#0**

**]IN# slot**

**]IN# slot [,A#]|A#**

**IN#**      e.g.,      **IN#2**      Immediate and deferred mode

**IN#**      e.g.,      **IN#3**      Immediate and deferred mode  
**IN#0**

**]OPEN pn [,S#] [,D#]**

**]OPEN pn [,L#] [,S#] [,D#]**

**OPEN**      e.g.,      **OPEN /MY/XMAS.LIST**      Deferred mode only

**OPEN**      e.g.,      **OPEN /MY/XMAS.LIST,S6,D2**  
**OPEN EXAMPLES,L28**  
**OPEN /MY/LIST,L200,S6,D1**

]READ pn [,F#] [,B#]

**READ**
 e.g.,  
 e.g.,  
 e.g.,  
 e.g.,

 READ /MY/XMAS.LIST  
 READ EXAMPLES,F4  
 READ /MY/ADDRESS.LIST,R5  
 READ LEDGER,R22,F3

Deferred mode only

]WRITE pn [,F#] [,B#]

]WRITE pn [,R#] [,F#] [,B#]

**WRITE**
 e.g.,  
 e.g.,  
 e.g.,  
 e.g.,

 WRITE /MY/XMAS.LIST  
 WRITE /MY/XMAS.LIST,F3  
 WRITE EXAMPLES  
 WRITE /MY/LIST,S6,D1

Deferred mode only

]CLOSE [pn]

**CLOSE**

e.g.,

 CLOSE  
 CLOSE /MY/XMAS.LIST

Immediate and deferred

]APPEND pn [,S#] [,D#]

]APPEND pn [,L#] [,S#] [,D#]

**APPEND**
 e.g.,  
 e.g.,

 APPEND /MY/XMAS.LIST  
 APPEND /MY/LIST/L200,D1  
 APPEND EXAMPLES,L28

Deferred mode only



]FLUSH [pn]

**FLUSH**      e.g.,      **FLUSH /MY/XMAS.LIST**      Immediate and deferred  
                                  **FLUSH MY/EXAMPLES**

]POSITION pn,F#

**POSITION**      e.g.,      **POSITION /MY/XMAS.LIST,F3**      Deferred mode only

]BLOAD pn [,A#] [,B#] [,L#|E#] [Ttype] [,S#] [,D#]

**BLOAD**      e.g.,      **BLOAD PIC1**      Immediate and deferred mode  
                                  **BLOAD PIC1,A\$4000**  
                                  **BLOAD PIC1,A\$4000,L\$2000**  
                                  **BLOAD PIC1,A16384,E\$24575**

]BSAVE pn ,A# ,L#|,E# [,B#] [,Ttype] [,S#] [,D#]

**BSAVE**      e.g.,      **BSAVE PIC1,A\$2000**      Immediate and deferred mode  
                                  **BSAVE PIC1,A8192**  
                                  **BSAVE PIC1,A\$2000,L\$2000**  
                                  **BSAVE PIC1,A8192,E\$4000**

]BRUN pn [,A#] [,B#] [,L#|,E#] [,S#] [,D#]

]— pn [,S#] [,D#]

**BRUN**      e.g.,      **BRUN TONE.BEEP**      Immediate and deferred mode  
                                  **BRUN BEEPER,A\$300**  
                                  **— TONE.BEEP**

]EXEC pn [,F#] [,S#] [,D#]

**EXEC**      e.g.,      **EXEC DUMMY**      Immediate and deferred mode

# APPENDIX D.

## ERROR MESSAGES

*Every great mistake has a halfway  
moment, a split second when it can  
be recalled and perhaps remedied.*

*Pearl S. Buck, 1943*

### OVERVIEW

The purpose of this appendix is to gather error messages from all sources together in one area so that they may be easily referenced.

When an error of any kind is detected by either Applesoft II BASIC, FILER, CONVERT, or ProDOS, the executing program is normally interrupted and the error message is displayed on the video display screen. The source of error messages may be determined by the format displayed. These are shown in Figure D.1.

<i>Source</i>	<i>Format</i>
Monitor	ERR
Integer BASIC	*** SYNTAX ERROR

Applesoft II BASIC	?SYNTAX ERROR
ProDOS	SYNTAX ERROR

Figure D.1. ERROR formats.

Notice that the source of error messages is indicated by the character, lack of characters, or multiple characters preceding the actual message.

## DETERMINING THE ERROR

When developing programs, errors may be handled in at least two different ways. First, it is recommended that during the actual development of programs you do no error trapping in code and let either Applesoft II BASIC or ProDOS intercept errors as they occur. In this way you will be able to identify and handle errors easily, making corrections as needed. The second way errors may be handled is through the ONERR GOTO or ONERR GOSUB capability of Applesoft II BASIC. These instructions allow you to write your own error-handling routines. Once you have committed yourself to handling your own errors, you must be prepared to handle all possible Applesoft II BASIC and ProDOS errors.

When an error does occur during program execution, the number of that error is stored in memory location 222 = \$DE. Therefore, you may use the following code to retrieve that error number.

```
JE = PEEK(222)
```

In this case, the variable E is assigned the value of the error that was stored in memory location 222.

Further, you may also determine the line of Applesoft II BASIC code being executed when the error occurred. This can be done using the code:

```
JLN = PEEK(218) + PEEK(219) * 256
```

This code will set the variable LN to the line number where the error occurred. By using the E and LN variables you are able to pinpoint errors quickly. By the way, the variables are not sacred; you may use any variable you choose.

## PRODOS MESSAGES

Figure D.2 shows you both the DOS 3.3 and ProDOS error code numbers, their meanings, and the possible causes.

<i>Code #</i>	<i>Message</i>	<i>Possible Causes</i>
1	LANGUAGE NOT AVAILABLE	Requested language not available (ProDOS only)
2 2	RANGE ERROR	Command option too small or too large
3	NO DEVICE CONNECTED	No device connected to the specified slot (ProDOS only)
4 4	WRITE PROTECTED	Write-protected tab on diskette
5 5	END OF DATA	Attempted to read beyond end of file or record
6 6	PATH NOT FOUND	No file with the indicated name
7	VOLUME MISMATCH	Wrong volume parameter (DOS only)
8 8	I/O ERROR	Disk drive door open, diskette not formatted, or wrong DOS
9 9	DISK FULL	Too many files on a diskette
10 10	FILE LOCKED	Attempt to write to a locked file
11	INVALID OPTION	Illegal option used with command (ProDOS only)
12 12	NO BUFFERS AVAILABLE	Memory is full, file cannot be opened
13 13	FILE TYPE MISMATCH	Wrong file type for the command being used or accessed
14 14	PROGRAM TOO LARGE	Program too large for the memory in your system
15 15	NOT DIRECT COMMAND	Illegal use of a command from the immediate mode
11 16	SYNTAX ERROR	Incorrect file name, option, comma, or pathname
17	DIRECTORY FULL	Directory already has 51 files recorded (ProDOS only)
18	FILE NOT OPEN	Attempted to access a closed file (ProDOS only)
19	DUPLICATE FILE NAME	File name already used with RENAME or CREATE command (ProDOS only)
20	FILE BUSY	File(s) already open (ProDOS only)
21	FILE(S) STILL OPEN	Previous program did not close file(s) (ProDOS only)

Figure D.2. DOS-ProDOS error codes.

The first two columns indicate the DOS 3.3 and ProDOS error numbers respectively.

## FILER AND CONVERT MESSAGES

This section lists the error messages presented by the FILER and CONVERT programs. The form of these messages is different from the Monitor, ProDOS, and Applesoft II BASIC versions. They are generated through the error trapping mechanism built into the programs by the programmer and not handled through the Monitor, Applesoft II BASIC, or ProDOS.

<i>Error message</i>	<i>FILER and CONVERT module</i>	<i>Comment</i>
CAN'T DELETE DIRECTORY FILE	Transfer (or List) Files	The name of the DOS 3.3 file you tried to transfer is also the name of a ProDOS directory file.
CAN'T TRANSFER DIRECTORY FILE	Transfer (or List) Files	This program does not allow directory files to be transferred.
DIRECTORY ALREADY EXISTS	Copy Files Make Directory	You tried to copy to a directory instead of a file, or you tried to create a directory with a name already present in the subdirectory or volume directory.
DIRECTORY EXPECTED	List ProDOS Directory Set Prefix	You entered a file name instead of a directory name.
DIRECTORY NOT EMPTY	DELETE FILES	You can only delete files, not directories that contain other files.
DIRECTORY NOT FOUND	List ProDOS Directory Copy Files Delete Files Compare Files Alter Write-Protection Rename Files Make Directory Set Prefix	The program cannot find the subdirectory you specified.

DISK II DRIVE TOO FAST	Copy a Volume Format a Volume	Your disk drive speed is too fast.
DISK II DRIVE TOO SLOW	Copy a Volume Format a Volume	Your disk drive speed is too slow.
DISK WRITE-PROTECTED	Copy a Volume Format a Volume Rename a Volume Copy Files Delete Files Alter Write-Protection Rename Files Make Directory Select Configuration Defaults Restore Configuration Defaults Transfer (or List) Files	You need to write something to the disk, but the disk has the write-protection notch covered. Be very careful when removing the tab.
DUPLICATE FILE NAME	Rename Files Transfer (or List) Files	You have tried to use a name that already exists or to transfer a file that already exists.
DUPLICATE VOLUME	Compare Volumes	Two diskette volumes have the same name
ERROR CODE = xxx	Copy Volumes where xxx is any code	This error could happen anytime, but probably will not due to the program error trapping routines.
FILES DO NOT MATCH	Compare Volumes	One or more of the bytes on the two volumes being compared do not match.
FILE EXPECTED	Delete Files Alter Write-Protection	You entered a volume directory instead of a file name or subdirectory.
FILE LOCKED	Delete Files Rename Files	You have tried to delete or rename a file that is write-protected.
FILE NOT FOUND	Copy Files Delete Files Compare Files Alter Write-Protection	The file does not exist in the directory specified.

	Rename Files Set Prefix Transfer (or List) Files Quit Copy Files	
FILE TOO LARGE		Not enough room exists on the diskette for the file you want to copy.
I/O ERROR	Trapped in all modules.	This error alerts you to the fact that you have an open drive door, empty disk drive, unformatted diskette, misaligned diskette, or bad blocks on the destination diskette.
ILLEGAL CHARACTER	Set Prefix Transfer (or List) Files	You used an illegal character in a pathname or file name.
ILLEGAL WILDCARD	Copy Files Delete Files Alter Write-Protection Rename Files List ProDOS Directory	You tried to use more than one wildcard per pathname
INSUFFICIENT MEMORY	During startup	Your system does not have 64K.
INVALID DATE	Set ProDOS Date	You entered an invalid date
INVALID DRIVE	Copy a Volume Format a Volume Rename a Volume Detect Bad Blocks Display Block Allocation Compare Volumes Select Configuration Defaults Change DOS 3.3 Slot and Drive Set Prefix	When you supply drive numbers, you may only use a 1 or 2. You entered an illegal drive number.
INVALID PATHNAME	Copy Files Delete Files	You used an illegal character in a pathname. The

	Alter Write-Protection Rename Files List ProDOS Directory Set Prefix Compare Files Select Configuration Defaults Restore Configuration Defaults	prefix may not be set correctly.
INVALID SLOT	Copy a Volume Format a Volume Rename a Volume Detect Bad Blocks Display Block Allocation Compare Volumes Select Configuration Defaults Change DOS 3.3 Slot and Drive	You selected a drive outside the allowable range of (1-7).
NAME TOO LONG	Set Prefix Set Prefix Transfer (or List) Files	The name you entered is longer than 15 characters.
NO DATA IN FILE	Transfer (or List) Files	There is no data in the file being transferred.
NO DEVICE CONNECTED	Format a Volume Copy a Volume Rename a Volume Detect Bad Blocks Display Block Allocation Compare Volumes Select Configuration Defaults Restore Configuration Defaults Change DOS 3.3 Slot and Drive Set Prefix	Disk drive is not connected to the slot specified or is not turned on.



NO DIRECTORY	List Volumes	The program did not find the volume specified.
NO PRINTER CONNECTED	List Volumes Detect Bad Blocks Compare Volumes List ProDOS Directory	You directed output to a printer, but the printer is not connected.
NO ROOM ON VOLUME	Transfer (or List) Files	The volume is full.
NOT A DOS 3.3 VOLUME	Transfer (or List) Files	The diskette specified with a slot-drive combination is not a DOS 3.3 diskette.
NOT A PRODOS DIRECTORY	Set Prefix	You did not specify a ProDOS directory.
NOT A PRODOS INTERPRETER	Quit	You did not enter a SYSTEM file name.
NOT A PRODOS VOLUME	Rename a Volume Display Block Allocation	You tried to use a ProDOS command with a non-ProDOS diskette.
NOT THE SAME DEVICE TYPE	Copy a Volume Compare Volumes Select Configuration Defaults	
NOT THE SAME DIRECTORY	Rename Files	You tried to name or rename a file or files into a new directory.
PATH NOT FOUND	Set Prefix	The volume was found, but the subdirectory was not found.
PATHNAME TOO LONG	Transfer (or List) Files Set Prefix	Pathname is either longer than 128 or 64 characters.
PATHNAMES INDICATE SAME FILE	Copy Files	Source and destination pathnames are the same.
PREFIX NOT SET	Transfer (or List) Files	You did not specify the proper prefix for the transfer.
SAME FIXED DISK	Copy a Volume Compare Volumes	You tried to copy a ProFile volume onto itself or compare it with itself.
VOLUME DIRECTORY FULL	Copy Files Make Directory	There is no more room in the directory for the files you want to add.

VOLUME FULL	Make Directory Copy Files	There is not enough room left on the diskette for the directory or file you want to add.
VOLUME NOT FOUND	Applies to almost all modules.	The program cannot find the specified volume name.
WILDCARD MUST BE FINAL NAME	Transfer (or List) Files	You entered a character after using the wildcard character.
WILDCARD NOT ALLOWED	Compare Files Make Directory Set Prefix	You used a wildcard character in a command that does not allow wildcards.
WILDCARD NOT PROCESSED	Rename Files	Pathname became too large when wildcard characters were substituted.
WILDCARD USE INCONSISTENT	Copy Files Rename Files	You did not use the wildcard in both source and destination pathnames.

Figure D.3. FILER and CONVERT error messages.

## APPLESOFT II MESSAGES

Applesoft II BASIC error messages are included here because these will also be seen when developing programs. It is very difficult to separate them and treat them individually, since both will occur as you develop your programs and program packages. However, your programs *will* treat them separately. They are here for your convenience.

<i>Code #</i>	<i>Message</i>	<i>Possible Causes</i>
0	NEXT WITHOUT FOR	Encountered a NEXT without first having executed a FOR statement.
16	SYNTAX ERROR	Instruction constructed incorrectly. Syntax is incorrect.
22	RETURN WITHOUT GOSUB	Encountered a GOSUB without first having executed a GOSUB statement.
42	OUT OF DATA	Attempted to READ beyond the end of available DATA.
53	ILLEGAL QUANTITY	Attempted to store information in an array beyond DIMension limits.
69	OVERFLOW	Number too large to be handled.

77	OUT OF MEMORY	Program and data are too large for the memory available.
90	UNDEFINED STATEMENT	Attempted to execute a statement that does not exist.
107	BAD SUBSCRIPT	Attempted to reference an element in an array that does not exist.
120	REDIMENSIONED ARRAY	Tried to dimension an array for the second time.
133	DIVISION BY ZERO	Tried to divide a number by the value of 0 or variable whose value is 0.
163	TYPE MISMATCH	Tried to assign a string to a number or vice versa.
176	STRING TOO LONG	String length is greater than 255 characters.
191	FORMULA TOO COMPLEX	Formula involves more than can be handled by language
224	UNDEFINED FUNCTION	Tried to execute a function that does not exist.
254	BAD RESPONSE TO INPUT STATEMENT	Entered the wrong response to the INPUT statement.
255	CTRL-C INTERRUPT ATTEMPTED	Entered to a CTRL-C to interrupt the currently executing program.
	CAN'T CONTINUE	Program cannot continue because an error has occurred. Language cannot identify error.
	ILLEGAL DIRECT COMMAND	Command cannot be used in the immediate mode

Figure D.4. Applesoft II BASIC messages.

# APPENDIX E.

## MISCELLANEOUS TOPICS

*Titles are but nicknames,  
and every nickname is a title.*

*Thomas Paine, 1791*

### OVERVIEW

The appendix provides a discussion of a number of miscellaneous topics that didn't seem to fit neatly elsewhere. There really isn't anything special about these topics, except that they may come in handy when you write your programs.

The topics covered briefly here are the clock/calendar card, the /RAM capability, and the ProFile hard disk.

### CLOCK/CALENDAR CARD

ProDOS has the capability of being able to support a clock/calendar card. This capability was not supported by DOS 3.3 directly. Now ProDOS directly supports a number of clock/calendar cards. In general, the following Applesoft II BASIC code may be used to activate and present the time on the video screen. Type in the following:

```

]NEW
]LIST

```

```

100 PRINT PR# 4: IN# 4
110 INPUT "%":T$
120 PR# 0: IN# 0
130 VTAB 2: HTAB 30: PRINT T$
140 END

```

<i>Lines</i>	<i>Description</i>
--------------	--------------------

Line 100	Activate the Clock/Calendar card located in slot 4. Allow for reading and writing to slot 4.
----------	---

Line 110	Input the time.
----------	-----------------

The % character signifies that you are using Applesoft II BASIC.

Notes: (1) The particular brand of clock/calendar card being used here uses the following special characters to determine the format of the time to be presented. These are:

“%” = Applesoft II BASIC

“&” = 24-hour clock display

“#” = Returns time in numeric format

“>” = Integer BASIC

“<” = 24 hour clock display

“,” = Interrupt rate is 64 Hz

“.” = Interrupt rate is 256 Hz

“/” = Interrupt rate is 2048Hz

(2) Please see your particular owners manual for specific code requirements.

Line 120	Return input and output to the video screen.
----------	--

Line 130	Position the cursor and print the time. Position to line 2 on the video screen. Move to horizontal position 30 on line 2. Print the time.
----------	--

Line 140	END the program.
----------	------------------

With a clock/calendar card installed you are able to time-date stamp a file for every update that is made and when a file is created. ProDOS will JSR = \$20 (jump to subroutine) to memory location 48902 = \$BF06 in hexadecimal. That memory location is the entry point to the DATETIME routine. If there is no routine installed then an RTS = \$60 will be stored at this location.

If you are using one of the Apple recommended clock/calendar cards, then when ProDOS finds the clock, it will set up the routine for you and place the correct jump address into the routine for you. If you are using some other clock/calendar card or one of your own making, you must write your own routines.

The routine should read the date and time from the card and place it in bytes 49040 through 49043 in the format shown in Figure E.1.

At this point, you have all that is necessary to write your own routine. Just make sure that the entry point and storage locations are maintained.

## **/RAM IN THE APPLE**

This capability was not supported by DOS 3.3. Now you are able to fully use a 16K add-on memory in slot 0 of Apple II and Apple II Plus. In addition, you may add an additional 64K to slot 0 for Apple II and Apple II Plus computers or an extended 80-column text card to the auxiliary slot of an Apple IIe.

When you boot up a ProDOS diskette, a check is made to determine the configuration of your particular system. If a 128K computer system is found, then the additional 64K bank of memory is designated as a RAM disk, /RAM, that is mapped physically as slot 3, drive 2. The reason for this is because normally an 80-column text card or any 80-column card resides in slot 3, drive 1. If you intend to use the additional 64K for any other purpose than a RAM diskette, you must be very careful and protect yourself from any possible data destructions.

For example, assume that you save the two high-resolution pages of auxiliary memory as /RAM/PAGE.1 and /RAM/PAGE.2. If this is done as the first two entries in the /RAM memory disk then PAGE.1 will be saved at memory locations \$2000 through \$3FFF and PAGE.2 will be saved at \$4000 through \$5FFF in the auxiliary memory space. These are actually "dummy" areas to be that you will use later.

However, there is really no specific formula for determining exactly where the blocks of /RAM will physically reside in the auxiliary memory. In addition, the logical blocks stored are not necessarily physically contiguous. As a result, there is not any guaranteed way to save specific fixed portions of any auxiliary memory space except through the procedure outlined above.

If you do wish to protect all of auxiliary memory that is not reserved by Apple Computer, Inc., then you will have to disconnect /RAM. In order to do that, there are three areas of main memory within the ProDOS image system global page that may be affected. These are:

- \$BF10-\$BF2F** These locations contain the disk device driver addresses.
- \$BF31** This location contains the number of devices minus 1.
- \$BF32-\$BF3F** These locations contain the list of disk device numbers.

These locations were shown and explained in Chapter 9.

In order to disconnect /RAM, the following steps need to be accomplished:

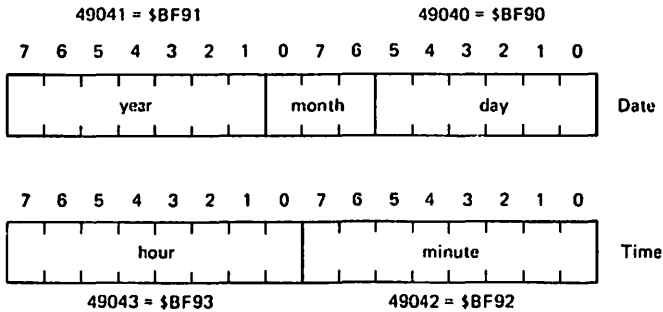


Figure E.1. Date and time locations.

1. Check the machine identification byte, \$BF98, to determine if you are operating a 128K system.
2. If you have a 128K system, then check:
  - a) Slot 3, drive 1 vector, \$BF16-\$BF17, to determine if "No Device Connected".
  - b) Slot 3, drive 2 vector, \$BF26-\$BF27, to determine if "No Device Connected".  
If true and equal to \$D0A2, then /RAM is already disconnected.
3. If you have determined that /RAM is on line, then you are ready to disconnect /RAM.
  - a) Retrieve the slot 3 device number. This is found in memory location \$BF26-\$BF27 or in the device list bytes starting at location \$BF32. You are looking for a \$FF = 255 value. You will want to save this value.
  - b) Remove the \$FF number from the device list.
  - c) Move any remaining device numbers up in the list.
  - d) Retrieve the slot 3, drive 2 device driver vector and save it for later re-installation.
  - e) Place the "No Device Connected" bytes, \$D0A2, in that vector location.
  - f) Decrement the device number count by one. This is the memory location \$BF31.

/RAM is now disconnected and you are free to use the unreserved areas for your own use.

If you are going to re-install /RAM, then you will have to essentially reverse the above procedure. After you have re-installed /RAM you will have to further re-install the /RAM directory.

## PROFILE HARD DISK STORAGE

With ProDOS you are now able to use the ProFile hard disk drive from Apple Computer, Inc. This gives you the ability to store up to an additional 5MB of auxiliary file storage.

One of the major problem areas encountered in providing an easily installed hard disk was the disparity between storage formats between DOS 3.3 with Applesoft II BASIC and Pascal with its storage format. ProDOS solves this disparity by making both storage media formats more consistent.

Therefore, you now have the ability to store Applesoft II BASIC programs, Pascal programs, binary programs, high-resolution memory pages, text files, Pascal data files, and other language files and data on the same media. In fact, you may intermix these various file types on the same directory or subdirectory.

Normally, the default slot for your hard disk is slot 5. This does not mean that it will not work in any other slot, but slot 5 is the recommended location for the ProFile hard disk.

Once you have your hard disk installed correctly, it may be referred to by slot 5, drive 1 or as the volume directory name, /PROFILE, pathname. With ProDOS, the ProFile may be addressed like any other disk storage device. Since you have 5 million bytes of storage, it is strongly recommended that you create a number of subdirectories for the storage of data.

The only real caution when working with ProFile is that you need to power up the ProFile first and wait until the READY light burns steady before powering up the rest of your system. This will take between 20 to 30 seconds before the ProFile is ready.

Once your ProFile is powered up and READY, power up the rest of your system with the ProDOS operating system. You are now ready to use pathnames to access files stored on ProFile. All commands and instructions for accessing files will now work correctly.



# APPENDIX F.

## PRODOS, APPLE III, AND SOS

*There's no sauce in the world*

*like hunger.*

*Cervantes, 1610*

### OVERVIEW

This appendix discusses the relationship of ProDOS and SOS, the operating system for the Apple III. SOS is pronounced “sauce.” The abbreviation stands for sophisticated operating system. This discussion should be helpful to those owning an Apple III, those contemplating upgrading their system to an Apple III, or software developers writing assembly-language programs for both machines.

### PRODOS AND SOS

In Chapter 9 it was explained that blocks 0 and 1 on a ProDOS diskette store the boot code. This code reads the operating system software and executes that code. What was not mentioned was that this boot code will run on either the Apple II computers or an Apple III.

As you power up either an Apple II or an Apple III computer system with a ProDOS diskette installed, the boot code read into memory and executed is loaded at 2048 = \$800.

The first item of business for this boot code is to determine whether the software is running on an Apple II or an Apple III computer system. Once this code makes that determination, then either:

- PRODOS, for the Apple II family
- SOS KERNEL, for the Apple III computer

will be loaded. If the correct file is not found for the computer being used, an error message is displayed.

The implication of this is that a single application may be written and placed on a single diskette that could be run on either computer. This single diskette could then be sold to owners of either Apple II computers or Apple III computers.

## FILE COMPARISONS

ProDOS and SOS use exactly the same directory structure on a diskette. Therefore, every file on a ProDOS diskette can be read by a SOS program and vice versa.

This means that you would have to use file types that are shared between the two systems. Look at the file type delineation in Section 3.3. Probably, there are only three file types that you will absolutely need. These are the directory (DIR), text (TXT), and binary (BIN). All of these are then able to share data between the two operating systems.

When you come across file types intended for the exclusive use of one particular system, but encountered on the other system, such as the CATALOG or CAT command, those file types are displayed with numbers rather than their type abbreviations. The FILER and CONVERT programs from ProDOS will recognize and display the names for all currently defined file types including SOS file types. ProDOS will display the file types in hexadecimal notation.

## SYSTEM COMPARISONS

The Apple III computers have a great deal more memory available than the Apple II computers. Because of this one overriding reason, the Apple III is able to have a much more complete and powerful operating system (SOS). SOS has very well defined and implemented:

- File Manager
- Device Manager
- Memory Manager
- Interrupt Handler
- Event Handler

ProDOS has the following:

- File Manager
- Memory Manager (Simplified)
- Memory Calls

The Apple III operating system has the ability to communicate with all peripheral devices such as consoles, printers, disk drives, or MODEMs. This is done by using the open, read, write, and close commands with the appropriate peripheral device. ProDOS only has the ability to manipulate files. ProDOS communicates with hardware devices through firmware driver code in ROM installed on the interface card in a slot. As it turns out, there is very little consistency in the way the communications protocols have been implemented.

The calls to ProDOS that were discussed in Chapter 9 are only a part of the calls that may be made to SOS. The calls that involve files are essentially the same. However, some of the differences are:

- ProDOS does not require that you specify the file size when you create the file, SOS does.
- SOS allows you to retrieve the file size in bytes. In ProDOS you must open the file and then issue a call in order to retrieve the file size.
- SOS uses a relative positioning in a file. ProDOS uses absolute positioning.

SOS has a very nice memory manager. When more memory is needed, SOS allocates the memory if more memory is available. When that requested memory is no longer needed, it is released to be used for the next request.

ProDOS allocates memory through its own memory manager. It finds memory by looking in the memory bit map. If a particular page is unmarked, then it may be allocated. When that page is no longer needed, it is released and the memory bit map is reset. In this way, protected areas of memory may be preserved.

SOS is able to honor interrupts from any device provided that particular device driver is installed and able to handle interrupts. By the way, the device driver and the interrupt handler are inseparable. ProDOS does not have device drivers. Interrupt handling routines must be installed separately. Further, SOS has an interrupt priority capability for each device in the system. ProDOS must poll the driver routines one by one until an interrupt is claimed.

# APPENDIX G. THE APA PROGRAM

*It is one of the beautiful compensations  
of this life that no one can sincerely try  
to help another without helping himself.*

*Charles Dudley Warner, 1873*

## OVERVIEW

The APA, Applesoft Programmer's Assistant, program is a binary program provided with your ProDOS diskettes to help you write programs. This program is a greatly expanded version of the RENUMBER program provided in DOS 3.3.

This appendix outlines the commands available with this program so that you will be able to take advantage of the program.

## STARTING APA

If you have your system already powered up then from the immediate mode type in the following command with the /EXAMPLES diskette in one of your disk drives.

## ] — /EXAMPLES/EXTRAS/APA

If your system is not powered up, then power up your system with any startup diskette. When you have your system up and running, then run the APA program using the — (DASH) command.

After APA has been loaded and executed, the following screen will be presented:

```
APPLESOFT PROGRAMMER'S ASSISTANT
VERSION 1.2
COPYRIGHT APPLE COMPUTER, 1979-83
```

]

When you have the Applesoft II BASIC prompt on the screen you are able to use any of the APA commands, write a program, run a program, or do anything else you want to do.

The next sections in this appendix list and explain the APA commands. The commands shown are shown in the same format as other commands were shown.

## AUTO COMMAND

This command allows you to enter programs faster and easier. It allows you to:

- Specify the starting line number.
- Specify the increment between line numbers.

The form of this command is:

]AUTO [st] [,inc]

<i>Option</i>	<i>Description</i>
---------------	--------------------

- |        |   |
|--------|---|
| [st]   | -This option specifies the starting number for code entry in a program. |
| [,inc] | -This option specifies the increment to be used between line numbers.   |

When you specify an increment with this command, that increment is activated by typing a carriage return followed by typing the space bar. For example, enter a line number followed by the program line. Then press the RETURN key and the space bar. This will give you the next incremented line number.

If you want a line number not in the increment sequence, just type the number you want in place of pressing the space bar.

*Examples of the AUTO Command*

AUTO 100            Start with line number 100.  
 AUTO ,10           Line increment is 10.  
 AUTO 1000,10      Start with line 1000 and increment by 10.

**MANUAL COMMAND**

If you want to leave the automatic line numbering mode, just execute the manual capability.  
 The form of this command is:

]MANUAL

All you need to do is type the command and you are returned to entering line numbers on your own.

**RENUMBER COMMAND**

This command is for the purpose of renumbering a portion or all of a program.  
 The form of this command is:

]RENUMBER st,inc,first,last

The RENUMBER capability is probably the most useful and powerful command in the APA program.

*Option    Description*

st	This is the starting line number of the code you want renumbered.
inc	This is the increment you want between line numbers.
first	This is the first line number of the present code that is to be renumbered.
last	This is the last line number of the present code that is to be renumbered.

Notice that there are no options in this command. They are all required in the command either by specifying them explicitly or by using the default values by using a comma. If any of the options are left out by using only a comma, then certain defaults are assumed by the RENUMBER command. These default values are:

starting line number	100
increment	10

first	0
last	63999

The RENUMBER command will not let you place one line number on top of another line number within any program.

#### *Examples of the RENUMBER Command*

```

RENUMBER    3000,10,300,400
RENUMBER    30000,,300,400
RENUMBER    10,,
RENUMBER    ,,300,
RENUMBER    ,,,

```

## **HOLD COMMAND**

The HOLD command is for the purpose of placing a program in memory on hold above HIMEM where it cannot be erased by loading another program into memory. In this way you may load a second program and then merge the two programs.

The form of this command is:

JHOLD

When this command is executed, the message "PROGRAM ON HOLD" is presented to you on the screen.

## **MERGE COMMAND**

This command is for the purpose of putting two programs or program segments together.

The form of this command is:

JMERGE

The general plan of attack in using this command is the following:

—First, load a program into memory.

JLOAD PROGRAM.1

—Now, place this program on hold.

## **]HOLD**

—Next, load a second program into memory.

## **]LOAD PROGRAM.2**

—At this point, you may renumber, edit, or anything else except run the second program. When you are ready, then merge the two programs.

## **]MERGE**

—At this point, both programs are put together into one program. You can now continue with the development of the program. By the way, this is exactly the way the XMAS.PROG in Chapter 10 was built, from numerous general subroutines stored on a subroutine diskette.

## **COMPRESS COMMAND**

This command is for the purpose of removing embedded remark statements from a program.

The form of this command is:

**]COMPRESS**

It is recommended that you save two versions of a program, the original with all of the remark statements and the compressed version with all of the remark statements removed. Aside: Before compressing a program, make sure your program does not branch to any statement that starts with REM.

## **SHOW COMMAND**

This command is for the purpose of displaying embedded control characters in a program.

The form of this command is:

**]SHOW**

Any control characters in a program will be displayed in a listing in inverse video.



## NOSHOW COMMAND

This command is for the purpose of turning off the SHOW capability. The form of this command is:

]NOSHOW

After using this command, any control characters will once again be invisible.

## LENGTH COMMAND

The purpose of this command is to calculate the current length of a program in memory. The form of this command is:

]LENGTH

When this command is executed, the length of a program is presented on the video screen in both decimal and hexadecimal.

## XREF COMMAND

This command will produce a cross-reference listing of the variables used in the Applesoft II BASIC program currently in memory.

The form of this command is:

]XREF

Once you execute this command, there will be a delay before you will actually get the cross-reference listing. The delay depends upon the length of the program in memory. All variables will be listed in alphabetical order with only the first two characters of the variable presented.

There are five kinds of variables identified in the cross-reference listing. These are:

<i>Type</i>	<i>Description</i>
\$	Represents a string variable.
(	Represents a real variable array.

\$ (	Represents a string variable array.
%	Represents an integer variable.
%(	Represents an integer variable array.

If your program is very long, then you may interrupt the listing by pressing the CTRL-S. To resume the listing use the CTRL-S again. The CTRL-S notation means to type the CTRL key and the S key together.

## CONVERT COMMAND

This command is for the purpose of converting a number from one form to another form. This command converts numbers from decimal notation to hexadecimal notation or vice versa.

The form of this command is:

**JCONVERT**

To use this command, use the following procedure:

**JCONVERT 255**

This will present the conversion on the screen:

**255 (\$00FF)**

In order to convert hexadecimal numbers, type:

**JCONVERT \$20**

This will return on the screen:

**32 (\$0020)**

## EXIT COMMAND

This command is for the purpose of exiting from the APA program.

The form of this command is:

**EXIT**

This causes your system to exit from the APA program and release all of the memory used by the program. If you want to use any of the APA commands later you will have to load and execute the program again.

# APPENDIX H.

## ASCII CHARACTER CODES

*I know sage, wormwood, and  
hyssop, but I can't smell  
character unless it stinks.*

*Edward Dahlberg, 1965*

DEC = ASCII decimal code  
HEX = ASCII hexadecimal code  
CHAR = ASCII character name  
n/a = not applicable

### Apple II Plus and Apple IIe

DEC	HEX	CHAR	WHAT TO TYPE	Meaning
0	00	NULL	ctrl @	Null character
1	01	SOH	CTRL A	Start of heading
2	02	STX	CTRL B	Start of text
3	03	ETX	CTRL C	End of text
4	04	EOT	CTRL D	End of transmission
5	05	ENQ	CTRL E	Enquiry
6	06	ACK	CTRL F	Acknowledge

## Apple II Plus and Apple IIe

DEC	HEX	CHAR	WHAT TO TYPE	
7	07	BEL	CTRL G	Bell
8	08	BS	CTRL H or <--	Back space
9	09	HT	CTRL I	Horizontal tab
10	0A	LF	CTRL J	Linefeed
11	0B	VT	CTRL K	Vertical tab
12	0C	FF	CTRL L	Formfeed
13	0D	CR	CTRL M or RETURN	Carriage return
14	0E	SO	CTRL N	Shift out
15	0F	SI	CTRL O	Shift in
16	10	DLE	CTRL P	Data link escape
17	11	DC1	CTRL P	Device control 1
18	12	DC2	CTRL R	Device control 2
19	13	DC3	CTRL S	Device control 3
20	14	DC4	CTRL T	Device control 4
21	15	NAK	CTRL U or -->	Negative acknowledgment
22	16	SYN	CTRL V	Synchronous idle
23	17	ETB	CTRL W	End of transmission block
24	18	CAN	CTRL X	Cancel
25	19	EM	CTRL Y	End of medium
26	1A	SUB	CTRL Z	Substitute
27	1B	ESC	ESCAPE	Escape
28	1C	FS	n/a	File separator
29	1D	GS	CTRL-SHIFT-M Apple II +	Group separator
30	1E	RS	CTRL ^	Record separator
31	1F	US	n/a	Unit separator
32	20		SPACE	
33	21	!	!	
34	22	“	“	
35	23	#	#	
36	24	\$	\$	
37	25	%	%	
38	26	&	&	
39	27	'	'	
40	28	(	(	
41	29	)	)	
42	2A	*	*	
43	2B	+	+	
44	2C	,	,	
45	2D	-	-	
46	2E	.	.	

## Apple II Plus and Apple IIe

<i>DEC</i>	<i>HEX</i>	<i>CHAR</i>	<i>WHAT TO TYPE</i>
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	:	;
59	3B	;	;
60	3C	<	<
61	3D	=	=
62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W

88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[	[ Apple IIe only
92	5C		Apple IIe only
93	5D	]	] Shift M Apple II +
94	5E	^	^
95	5F	—	— n/a Apple II +

## Apple IIe only

<i>DEC</i>	<i>HEX</i>	<i>CHAR</i>	<i>WHAT TO TYPE</i>
96	60	`	`
97	61	a	a
98	62	b	b
99	63	c	c
100	64	d	d
101	65	e	e
102	66	f	f
103	67	g	g
104	68	h	h
105	69	i	i
106	6A	j	j
107	6B	k	k
108	6C	l	l
109	6D	m	m
110	6E	n	n
111	6F	o	o
112	70	p	p
113	71	q	q
114	72	r	r
115	73	s	s
116	74	t	t
117	75	u	u
118	76	v	v
119	77	w	w
120	78	x	x
121	79	y	y
122	7A	z	z
123	7B	{	{
124	7C		
125	7D	}	}
126	7E	~	~
127	7F	DEL	DEL

# APPENDIX I. DECIMAL TO HEXADECIMAL CONVERSIONS

*A wise man changes his mind,  
a fool never will.*

*Spanish Proverb*

The top row represents the decimal digits from 0 through 9 with a unit's value.

The left column of numbers represents the decimal digits zero through 25 with a 10's value.



	0	1	2	3	4	5	6	7	8	9
0	00	01	02	03	04	05	06	07	08	09
1	0A	0B	0C	0D	0E	0F	10	11	12	13
2	14	15	16	17	18	19	1A	1B	1C	1D
3	1E	1F	20	21	22	23	24	25	26	27
4	28	29	2A	2B	2C	2D	2E	2F	30	31
5	32	33	34	35	36	37	38	39	3A	3B
6	3C	3D	3E	3F	40	41	42	43	44	45
7	46	47	48	49	4A	4B	4C	4D	4E	4F
8	50	51	52	53	54	55	56	57	58	59
9	5A	5B	5C	5D	5E	5F	60	61	62	63
10	64	65	66	67	68	69	6A	6B	6C	6D
11	6E	6F	70	71	72	73	74	75	76	77
12	78	79	7A	7B	7C	7D	7E	7F	80	81
13	82	83	84	85	86	87	88	89	8A	8B
14	8C	8D	8E	8F	90	91	92	93	94	95
15	96	97	98	99	9A	9B	9C	9D	9E	9F
16	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9
17	AA	AB	AC	AD	AE	AF	B0	B1	B2	B3
18	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD
19	BE	BF	C0	C1	C2	C3	C4	C5	C6	C7
20	C8	C9	CA	CB	CC	CD	CE	CF	D0	D1
21	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB
22	DC	DD	DE	DF	E0	E1	E2	E3	E4	E5
23	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
24	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9
25	FA	FB	FC	FD	FE	FF	100			

This means that the hexadecimal equivalent of the decimal value of 32 can be found in the fourth row,  $3 \times 10 = 30$ , plus the third column,  $2 \times 1 = 2$ . This is a 32 decimal, which equals a \$20 hexadecimal.

# BIBLIOGRAPHY

*A sick man that gets talking about himself,  
a woman that gets talking about her baby,  
and an author that begins reading out of  
his own book never knows when to stop.*

*Oliver Wendell Holmes, 1872*

## A

Ahl, David H., *101 Basic Computer Programs* (Digital Equipment Corporation, 1974).

Apple Computer, Inc. Publications (Apple Computer, Inc., 1977-1981).

<i>Apple II Monitors Peeled</i>	D2L0013
<i>Apple II Reference Manual</i>	A2L0001A
<i>Applesoft II Manual</i>	A2L0006
<i>Autostart ROM</i>	A2L0022
<i>Programmer's Aid</i>	A2L0011
<i>The Applesoft Tutorial</i>	A2L0018
<i>Parallel Printer Manual</i>	A2L0004
<i>Apple II Basic Manual</i>	A2L0005
<i>Communications Manual</i>	A2L0007
<i>The DOS Manual (DOS 3.2)</i>	A2L0012
<i>The DOS Manual (DOS 3.3)</i>	A2L0036
<i>Silentype Printer Manual</i>	A2L0034

<i>Graphics Tablet Manual</i>	A2L0033
<i>Apple 6502 Assembler/Editor</i>	A2L0039
<i>Applesoft Tool Kit</i>	A2L0038
<i>Apple IIe Owner's Manual</i>	A2L2001
<i>Apple IIe Applesoft Tutorial</i>	A2L2003
<i>Applesoft Reference Manual</i>	
Volume 1	A2L2005
Volume 2	A2L2005
<i>Apple IIe 80-column Text Card</i>	A2L2006
<i>ProDOS USER'S Manual</i>	
<i>ProDOS Technical Reference Manual</i>	A2L2015
<i>Basic Programming with ProDOS</i>	

*Apple Orchard* (Dilithium Press, 1981).

Artwick, Bruce, *Microcomputer Interfacing* (Prentice-Hall, Inc. 1980).

## B

Blackwood, Brian D. and Blackwood, George H., *Applesoft Language* (Howard Sams & Co., Inc. 1981).

Bryan, Paul, *Programming Your Apple II Computer* (Tab Books, Inc. 1982).

*BYTE Magazine* (Various Issues)

## C

*CALL A.P.P.L.E. Magazine*

Campbell, J. L. and Zimmerman, Lance, *PROGRAMMING the APPLE: A Structured Approach* (Robert J. Brady Co., 1982).

Campbell, J. L., *Tips and Techniques for the Apple II Plus/Apple IIe* (Robert J. Brady Co., 1984).

Carnahan, Luther and Wilkes, *Applied Numerical Methods* (John Wiley & Sons, 1969).

Coan, James S., *Advanced Basic* (Hayden Book Company, Inc. 1977).

Coan, James S., *Basic Basic* (Hayden Book Company, Inc. 1978).

Conte & DeBoor, *Elementary Numerical Analysis: An Algorithmic Approach* (McGraw-Hill Book Company, 1965).

Conway, Richard and Gries, David, *An Introduction to Programming* (Winthrop Publishers, Inc. 1975).

## D

Donovan, John J., *Systems Programming* (McGraw-Hill publishers, 1972).

Dyckman, Thomas R. and Thomas, Joseph L., *Algebra and Calculus for Business* (Prentice-Hall, Inc., 1974).

## E

Espinoza, Christopher, *Apple II Reference Manual* (Apple Computer, Inc., 1979).

F

- Findley, Robert, *6502 Software Cookbook* (Scelbi Publications, 1979).  
Finkel, LeRoy and Brown, Jerald R., *Apple BASIC: Data File Programming* (John Wiley & Sons, Inc., 1982).

G

- Gilder, Jules, *Basic Computer Programs in Science and Engineering* (Hayden Book Company, 1980).  
Gildersleeve, Thomas R., *Organizing and Documenting Data Processing Information* (Hayden Book Company, 1977).  
Goldsmith, W. B., Jr., *BASIC Programs for Home Financial Management* (Prentice-Hall, Inc., 1981).  
Grillo, John P., *Introduction to Graphics* (William C. Brown Publishers, 1981).

H

- Hornbeck, R. W., *Numerical Methods* (Quantum Publishers, Inc., 1975).  
Hyde, Randy, *Using 6502 Assembly Language* (DATAMOST Inc., 1981).

I

- Inman, Don and Inman, Kurt, *Apple Machine Language* (Reston Publishing Company, Inc., 1981).  
*Interface Age Magazine* (Various Issues).

K

- Kemeny, J. G. and Kurtz, T. E., *BASIC Programming* (John Wiley & Sons, Inc., 1980).  
Ketter & Prawel, *Modern Methods of Engineering Computation* (McGraw-Hill Book Company, 1969).  
*KILOBAUD Magazine* (Various Issues).  
Knuth, Donald E., *The Art of Computer Programming*, Vol. 1 (Addison Wesley, 1968).  
Knuth, Donald E., *The Art of Computer Programming*, Vol. 2 (Addison Wesley, 1969).  
Knuth, Donald E., *The Art of Computer Programming*, Vol. 3 (Addison Wesley, 1973).

L

- Lien, David A., *The Basic Handbook* (COMPUSOFT PUBLISHING, 1979).  
Luebbert, William F., *What's Where in the APPLE?* (MICRO INK, Inc., 1981).

M

- McGowan, Clement L. and Kelly, John R., *Top-Down Structured Programming Techniques* (Petrocelli/Charter Publishing, 1976).  
*MICRO Magazine* (Various Issues).

## N

- Nagrin, Paul and Ledgard, Henry F., *BASIC with Style* (Hayden Book Company, Inc., 1978).  
 Nazem, S. G., *The Folks Who Brought You Apple*, (*FORTUNE Magazine*, January 1981).  
*NIBBLE Magazine* (Various Issues).

## P

- PEEKing at Call A.P.P.L.E.* (Apple Pugetsound Library Exchange, 1979).  
*PEELINGS Magazine* (Various Issues).  
 Perles, Benjamin and Sullivan, Charles, *Modern Business Statistics* (Prentice-Hall, Inc.).  
*Personal Computing Magazine* (Various Issues).  
 Pooch, Udo W. and Chattergy, Rahul, *Designing Microcomputer Systems* (Hayden Book Company, Inc., 1979).  
 Poole, Lon and Borchers, Mary, *Some Common Basic Programs* (OSBORNE/McGraw-Hill, 1979).  
 Poole, Lon and McNiff, Martin, and Cook, Steven, *Apple II User's Guide* (OSBORNE/McGraw-Hill, 1981).

## R

- Raskin, Jeff, *Apple II Basic Programming Manual* (Apple Computer, Inc., 1978).  
 Richardson, Caryl, *The Applesoft Tutorial* (Apple Computer, Inc., 1979).

## S

- Simon, David E., *BASIC from the Ground Up* (Hayden Book Company, 1978).  
 Sippl, Charles J., *Microcomputer Dictionary and Guide* (MATRIX Publishers Inc., 1976).  
 Spencer, Donald D., *Game Playing with BASIC* (Hayden Book Company, Inc., 1975).  
 Stanton, Jeffrey, *Apple Graphics & Arcade Game Design* (The Book Co., 1982).  
 Sternberg, Charles D., *BASIC Computer Programs for the Home* (Hayden Book Company, Inc., 1980).  
 Sternberg, Charles D., *BASIC Computer Programs for Business* (Hayden Book Company, 1980).

## W

- Wadsworth, Nat, *Introduction to Low Resolution GRAPHICS* (Scelbi Publications, 1979).  
 Wadsworth, Nat, *Graphics Cookbook for the Apple Computer* (Scelbi Publications, 1980).  
 Wagner, Roger, *Assembly Lines: The Book* (A SOFTALK BOOK, 1982).  
 Weston, J. F., *Managerial Finance* (The Dryden Press, 1972).  
 Worth, Don and Lechner, Pieter, *Beneath Apple DOS* (Quality Software, 1981).

# GLOSSARY

*Words in their primary or immediate  
signification, stand for nothing but the  
ideas in the mind of who uses them.*

*John Locke, 1690*

**access time.** The time required to locate and read or write data on a direct access storage device, such as the disk drive.

**acoustic coupler.** Hardware designed to connect a telephone handset to a computer system.

**address.** 1. A numeric location of data, usually in memory. This is normally expressed as a hexadecimal number. 2. A diskette address is normally expressed in terms of the track and sector numbers.

**algorithm.** A prescribed set of rules or processes for the solution of a problem in a finite number of steps.

**alphanumeric.** A term that refers to the class of characters that includes all of the characters of the alphabet and arabic numerals.

**Apple IIe.** A personal computer in the Apple II family, manufactured and sold by Apple Computer, Inc.

**Applesoft II BASIC.** An extended version of the BASIC programming language used with the Apple II Plus or Apple IIe computer. An interpreter for creating and executing programs.

**AND.** The logical operator that describes the inclusive set.

**APPEND.** Attach to the end of. The APPEND command is used to write new data to the end of an existing file.

**argument.** An independent variable.

**array.** An orderly arrangement of storage elements having both dimension and value.

**ASCII.** An acronym for the American Standard Code for Information Interchange. This is normally the hexadecimal-to-character conversion code.

**assembly language.** This refers to a language that uses mnemonic symbols that relate on a nearly one-to-one correlation to the native symbols of the individual computer.

**asynchronous.** Signals or device actions that are not synchronized to a master clock frequency.

**auxiliary slot.** The special expansion slot inside the Apple IIe used for the 80-column card and the memory extension.

**back up.** Make a copy of a program, data, or diskette.

**BASIC.** Beginners All-purpose Symbolic Instruction Code. A common, high-level programming language. It was developed by Kurtz and Kemeny at Dartmouth College in 1963 and has become the most popular microcomputer language.

**binary.** The representation of numbers in terms of powers of two. A term used to describe the base two number system.

**binary file.** 1. A file whose data is to be interpreted in its binary form. 2. A file of the BIN type.

**bit.** A binary digit in the binary number system. This is the smallest possible unit of information consisting of a simple two-way choice.

**BLOAD.** Binary load.

**block.** 512 bytes of data. This is the unit of storage used by ProDOS.

**BLOCKS.** When you use the CAT or CATALOG command, the column on the screen labeled BLOCKS lists the number of blocks of disk space occupied by the file in that directory.

**boot.** The process of getting a computer system powered up and running.

**boot disk.** A disk that contains all of the information needed to get the computer system running.

**BPS.** Bits Per Second. A common measure of the rate of flow of information between digital systems.

**branch.** This refers to a departure from the normal sequential order of processing.

**BRUN.** Binary run. The BRUN command causes a binary program to be loaded into memory and run.

**BSAVE.** Binary save. The BSAVE command causes the binary data in some portion of memory to be saved as a disk file.

**buffer.** An area in memory that stores data temporarily.

**bug.** A man-made error in hardware or software. An error in a program that causes the program not to work as intended.

**byte.** That 8-bit unit of data or information considered a word in microcomputers. This is the smallest unit of information in memory that may be addressed.

**call.** A BASIC instruction on the Apple II Plus or Apple IIe that invokes a machine-language subroutine.

**CAT.** This command causes a list of the names and characteristics of all files in a directory to be displayed.

**CATALOG.** A list of all files stored on a diskette. See CAT.

**CHAIN.** The CHAIN command runs a BASIC program without first erasing the variables in memory.

**character.** Any symbol that has meaning to people. Normally thought of as being one of the ASCII set of characters.

**checksum.** A character residing at the end of a data block. Used for error checking.

**clock.** The most basic source of synchronizing signals in electronic equipment. The generator of periodic electrical pulses that control the timing of switching circuits in microprocessors.

**CLOSE.** This command is issued when you have finished using a file.

**code.** 1. A method of representing something in terms of something else; e.g., the code used to program in the BASIC language. 2. A number or symbol used to represent some piece of information in a compact or easily processed form. 3. The statements or instructions making up a program.

**coding.** The conversion of flowcharts into a particular computer language.

**command.** 1. The portion of an instruction that specifies the operation to be performed. 2. A communication from the user to a computer system directing it to perform some immediate action.

**compiler.** A program that translates a high-level language into machine-readable code.

**computer.** 1. That arrangement of electronic elements necessary to solve problems. 2. An

- electronic device for performing predefined computations at high speed and with great accuracy.
- constant.** A data item that remains unchanged during the running of a program.
- CPU.** Central Processing Unit.
- CREATE.** This command creates a new file. It places a new file of a designated type into a designated directory.
- CRT.** Cathode Ray Tube.
- cursor.** 1. A marker or symbol that delineates where the next action will take place. 2. The blinking square of white on a black screen when in the NORMAL mode on an Apple II Plus. 3. The blinking checkerboard square of white on a black screen on an Apple IIe.
- DASH (—).** This command runs a basic, machine language, EXEC, or interpreter program.
- data.** A general term that is used to denote any or all information, facts, numbers, letters, or symbols, which can be processed or produced by a computer.
- DCT.** Device Characteristics Table; describes the hardware characteristics of the disk drive.
- debug.** To detect, locate, and correct errors in a program.
- decision.** The operation performed by the computer that enables it to choose between alternatives.
- decrement.** The fixed amount that is subtracted from another quantity.
- DELETE.** The command that removes a file from its directory.
- dellimiters.** Characters that limit the size, range, or limits of some physical or logical element of a computer system.
- directory file.** A file that contains the names and locations on the disk of other files.
- disk.** Name of a magnetic storage peripheral device.
- diskette.** Magnetic storage media of varying size used with disk drives.
- DMA.** Direct Memory Access. The procedure of bypassing the CPU and moving data directly to or from memory.
- documentation.** The written explanation of a program.
- driver.** The program or program segment that controls the execution of other program modules.
- ENDFILE.** End of file.
- EOF.** End Of File.
- EPROM.** Electrically Programmable Read Only Memory.
- error.** A mistake.
- EXEC.** This command causes input to be taken from a sequential text file rather than from the keyboard.
- execution time.** Time required by a microprocessor to execute an instruction in a high-level language, a program, or a machine language instruction.
- field.** In a file, a string of characters preceded by a carriage return character and terminated by a carriage return character.
- file.** A named collection of data or a program normally stored on a mass media, usually a disk drive.
- file name.** The name that identifies a file.
- file type.** The type of data that is stored in the file.
- firmware.** Programs stored in a ROM chip or chips.
- flag.** Information that indicates whether a particular condition is present or not. See **switch**.
- FLUSH.** Send unwritten data to its file.
- format.** 1. A predetermined arrangement of data. 2. The form in which information is organized or presented. 3. To prepare a blank diskette to receive information by dividing its surface into tracks and sectors.
- full duplex.** The ability to communicate in two directions simultaneously.
- graphics.** 1. A system that displays characters or figures in some form on a display tube. 2. Information presented in the form of pictures or images. 3. The display of pictures or images on a computer display screen.
- half duplex.** Bidirectional communications in which data flows in one direction only at any one time.



**handshake.** Exchange of predetermined signals when a connection is established between two data set devices.

**hard copy.** Typewritten or printed characters produced on paper by a computer.

**hardware.** The metallic, plastic, or other "hard" components of a computer system.

**HELPSCREENS.** A file, stored on the /PRODOS diskette, which contains all of the help screens.

**hertz.** A unit of frequency equal to one cycle per second.

**hexadecimal.** A numbering system that is based on the powers, radix, of 16. It uses the symbols 0-9 and A-F.

**high-level language.** A problem-oriented programming language (e.g., BASIC), closely related to a natural language such as English, that does not reflect the nature of machine language.

**high order.** The most significant bit in a byte or the most significant byte in a word.

**HLL.** High-Level Language.

**IN#.** This command designates the source of subsequent input characters.

**increment.** The fixed amount that is added to another quantity.

**input.** 1. A device used for bringing data into another device. 2. In the English language, the "input" may be used as a noun instead of "input data," "input signal," etc., when the usage is clear. 3. In a computer language, "input" refers to the command for the collection of data from some input device.

**instruction.** The statement written to the computer.

**integer.** A number that does not have a decimal point.

**integration.** In computer terminology, the accumulation of a large number of circuits (say 1,000 or more) on a single chip or substrate.

**interactive.** The condition in which the operator and the computer are in direct communication during the execution of a program.

**interface.** 1. The common boundary that matches adjacent components in a computer system. 2. The devices, rules, or conventions

by which one component of a system communicates with another.

**interpreter.** A program that operates directly on a source program in memory. The interpreter translates the instructions of the source program one by one and executes them immediately.

**jump.** Analogous to the GOTO statement in an HLL. Perform a hard branch.

**K.** Abbreviation for kilo.

**kilo.** In engineering, 1,000. In computer terminology, 1024.

**kilobyte.** 1024 bytes.

**language.** The set of rules or conventions required to write computer programs.

**language card.** An Apple computer interface card which, when placed in slot zero of an Apple II or Apple II Plus with 48K memory, adds 16K more memory to the computer.

**line number.** The positive integer that will begin each BASIC program statement.

**link.** An address pointer as an element of a linked list of data.

**list.** A one-dimensional sequential array of data items.

**literal.** The sequence of characters enclosed within quotation marks. See **string**.

**LOAD.** This command brings a BASIC program into memory from a file.

**LOCK.** This command protects a file from being accidentally renamed, deleted, or altered.

**logic.** In a computer program, the systematic scheme that defines the interactions of various components of a computer program.

**loop.** A sequence of instructions cyclically repeated a number of times. This is normally a construction in code.

**lsb.** Least significant bit.

**LSB.** Least Significant Byte.

**LSI.** Large-Scale Integration, a component density of more than 100 per chip.

**M.** Mega or million.

**machine language interface.** The set of machine language routines, stored in the file named

**/PRODOS**. with which ProDOS talks to disk drives.

**memory**. 1. One of the basic components of a computer. 2. The main storage unit of the computer system.

**menu**. 1. A table of items from which selections are made that determine the order of execution of program components. 2. A list of choices presented by a program.

**micro**. A small, usually 8-bit machine. Slang for microcomputer.

**microcomputer**. Complete computer of the micro size.

**microprocessor**. CPU built into chips by means of LSI technology.

**MODEM**. MODulator-DEModulator.

**module**. A section of code that performs only a single function or set of functions.

**monitor**. 1. A CRT that does not contain an RF section. 2. A machine language program that resides in memory and performs basic functions.

**motherboard**. Name for the main board on the Apple.

**msb**. Most significant bit.

**MSB**. Most Significant Byte.

**NAME**. When a catalog of files is displayed on the screen, the NAME column contains the names of the files in the listed directory.

**NMOS**. N-channel MOS circuit.

**numeric**. A data type that is numeric in nature. Refers to numbers.

**Nybble**. Name given to one-half of a byte.

**OPEN**. This command allocates space in memory for a file's buffers, and sets the file position pointer to the beginning of the file.

**operand**. This is a data item on which some operation is performed.

**operating system**. A machine language program that manages a multiplicity of functions in a computer system, including peripherals.

**operator**. 1. The action taken upon an operand. 2. The person operating the computer.

**option**. An item in the syntax of a ProDOS command that determines a single aspect of the computer's action.

**OR**. A logical operator.

**output**. 1. The display of the final computer solution on some visual display; e.g., CRT or printer. 2. Information transferred from a computer to some external destination.

**page**. A group of 256 bytes of memory that shares the same high order address byte.

**pagination**. Page numbering and page formatting in a report.

**parallel**. The simultaneous transmission of, storage of, or logical operation on, a word in the computer memory.

**parameter**. Variable that can take the place of one or more other variables in mathematics.

**parity**. The odd-even characteristic of 1's in a byte. This scheme allows for the detection of errors.

**parse**. To take apart and analyze a word, sentence, instruction, etc.

**pathname**. A series of file names, preceded and separated by slashes, that indicates the entire path from the volume directory to the file.

**peripheral**. Auxiliary equipment used in the computer system that is external to the computer itself.

**pixel**. The smallest addressable picture element that may be displayed on a video screen.

**pointer**. An address or number of a block of data. The address "points" to the data.

**POSITION**. This command causes a specified number of fields to be read and discarded from an open file.

**PR#**. This command sends output to the slot specified.

**prefix**. A settable pathname that indicates a directory file.

**/PRODOS**. The volume name of the disk that contains the ProDOS program.

**ProDOS command**. Any one of the 28 commands recognized by ProDOS.

**program**. 1. The set of statements or instructions that tells the computer how to solve a particular problem. The program MUST be stored in memory. 2. A set of instructions describing the actions for a computer to perform.

**PROM**. Programmable Read Only Memory.

**prompt.** An output string that lets the operator know what is required. A character that informs the operator that some input is required.

**RAM.** Random Access Memory. Access time is time independent of the physical location of memory accessed.

**/RAM.** The volume name of a small volume automatically placed by ProDOS in the alternate 64K of an Apple IIe when the 80-column extended text card is installed.

**READ.** This command, when used with the OPEN command, prepares a file to be read.

**record.** A collection of associated data items or fields.

**register.** Fast temporary-storage locations, usually inside the microprocessor itself.

**RENAME.** This command allows you to change the name in the file.

**resolution.** Refers to the size of a picture element (pixel) displayed on a video screen.

**RESTORE.** This command clears the BASIC variables currently in memory, and it then reads in a new set of variables from a variable file.

**ROM.** Read Only Memory. This information may be accessed by the microprocessor, but may not be changed.

**RUN.** 1. To execute a program. 2. To load and execute a program from some external device.

**RWTS.** Read/Write/Track/Sector. A collection of subroutines that allows access to the diskette at the track/sector level.

**SAVE.** This command lets you save the BASIC currently in memory.

**scroll.** The act of moving the text display, usually upward, to make room for information to be displayed on the screen.

**sector.** The smallest addressable unit on a diskette track that may be changed.

**semantic.** The meaning of an instruction.

**serial.** The time-sequential transmission of, storage of, or logical operations on, a word in a computer.

**sequential.** A mode of data retrieval in which each data element is read in exactly the same order in which it was written.

**simplex.** Ability to communicate in one direction only.

**software.** The programs, languages, and operating procedures of a computer system.

**source code.** A program that is in a form that is understandable to humans.

**statement.** Instructions entered into the computer memory. In BASIC, they are numbered.

**STORE.** This command causes the BASIC variables currently in memory to be stored in a variable file on the diskette.

**string.** A group of characters that make up a single unit.

**subroutine.** A series of computer instructions that may be branched to, executed, and know where to return upon completion.

**switch.** Information that is either on (set) or off (reset). A switch has only two allowable states. See flag.

**syntax.** The correctness of the written instruction.

**system.** A file with a name of the form XXX.SYSTEM must be in the volume directory of every boot diskette.

**table.** An orderly arrangement of data, usually in two dimensions.

**text file.** A file whose contents are interpreted as characters encoded using the ASCII format.

**trap.** To catch potential errors, usually done using error handling routines.

**TYPE.** In a catalog, the column with this heading names the type of each file listed.

**unlock.** This command reverses the effect of the LOCK command.

**/UTILITIES.** This disk contains utilities programs for your use.

**variable.** A symbolic quantity that can assume any of a given set of values. Provides storage in memory for either numbers or strings.

**vector.** A one dimensional array of numbers or characters. A collection of pointers.

**volume.** A source or destination of information.

**word.** That group of characters in memory, operated on by the computer. In the case of microcomputers, this is usually a byte.

**WRITE.** This command, when used after an **OPEN** command, prepares a file to be written to.

**zero page.** The first 256 memory locations in a 6502 processor. This area has the address range of \$0000-\$00FF.

# INDEX

- ALLOC INTERRUPT call, for MLI 173, 174
- ALTER WRITE-PROTECTION 121
- APA program commands 274-281
  - AUTO 275
  - COMPRESS 278
  - CONVERT 280
  - EXIT 280-281
  - HOLD 277
  - LENGTH 279
  - MANUAL 276
  - MERGE 277-278
  - NOSHOW 279
  - RENUMBER 276-277
  - SHOW 278
  - XREF 279-280
- APPEND command 64, 76-77, 81, 238, 254
  - used with random-access file 81, 93-94
  - used with sequential-access file 64, 76-77
- Apple II computers, operating systems for 230
- Applesoft Programmer's Assistant. *See* APA program
- Applesoft II BASIC 5, 182
- ASCII character codes 282-285
- AUTO command 275-276
- AUTOSTART 4
- AUTOSTART ROM chip program 28
- BASIC. SYSTEM program 27, 29
- Binary addresses 98
- Binary disk file 102
- Binary files 96-105
  - addresses 98
  - command options 98
- Binary program 96
- BLOAD command 97, 99-101, 239, 255
- Block 126
- BLOCK ALLOCATION 132
- Block File Manager (BFM) 152-153
- BNE 152
- Boot 8
- BRUN command 97, 102-103, 239, 255
- BSAVE command 97, 101-102, 239, 255
- BSC 152
- Carriage return character 65, 66, 74, 75
- CAT command 26, 32-34, 47-49, 233, 251

- CATALOG command 26, 32-34, 47-49, 233, 251
- CHAIN command 57, 235, 252
- Clock/calendar card 266-268
- CLOSE command 237, 254
  - for MLI 167
  - used with random-access file 81, 89, 92
  - used with sequential-access file 64, 76
- CNUM 152
- Commands, APA program 274-280
- Commands, DOS 3.3 not supported by ProDOS 231
- Commands, ProDOS
  - APPEND 64, 76-77, 81, 93-94, 238, 254
  - BLOAD 97, 99-101, 239, 255
  - BRUN 97, 102-103, 239, 255
  - BSAVE 97, 101-102, 239, 255
  - CAT 26, 32-34, 47-48, 233, 251
  - CATALOG 26, 32-34, 47-49, 233, 251
  - CHAIN 57, 235, 252
  - CLOSE 64, 70, 81, 89, 92, 167, 237, 254
  - CREATE 46, 47, 52-54, 154-155, 234, 252
  - DELETE 55, 81, 93, 234, 252
  - EXEC 107, 110-111, 240, 255
  - FLUSH 65, 77-78, 81, 89, 238, 255
  - HELP 21-22, 231, 250
  - IN# 60-61, 97, 103-104, 236, 253
  - LOAD 26, 42-43, 232, 251
  - LOCK 55-57, 234, 252
  - NOHELP 22, 232, 250-251
  - OPEN 64, 72-74, 89-90, 161-164, 236, 253
  - POSITION 65, 78, 238, 255
  - PR# 59-60, 97, 103-104, 235, 253
  - PREFIX 37-39, 47, 49-52, 233, 251-252
  - READ 64, 74-75, 81, 90-91, 165-166, 237, 254
  - RENAME 54-55, 156, 234, 252
  - RESTORE 57-59, 235, 253
  - RUN 26, 40-42, 232, 251
  - SAVE 26, 43, 233
  - STORE 57, 58, 234, 252
  - UNLOCK 55-57, 234, 252
  - WRITE 64, 75, 81, 91-92, 237, 254
- COMPARE FILES 121
- COMPARE VOLUMES 133-134
- COMPRESS command 278
- CONVERT command 280
- CONVERT program 23, 112-115, 136-140
  - keystrokes for 113-114
  - uses 136-139
- COPY FILES 119-120
- COPY A VOLUME 127-128
- CREATE command 46, 47, 52-54, 234, 252
  - for MLI 154-155
- Creation program 64
  - (DASH) command 8-9, 39-40
  - used for binary program 96
- DB 152
- DEALLOC INTERRUPT call, for MLI 173-174
- Debugging phase 183-184
- Decimal-hexadecimal conversions 286-287
- DELETE command 55, 81, 234, 252
  - used with random-access file 93
- Delimiters 11
- Designing a program 181-185
  - confirming correctness 183-185
  - defining problem 182
  - describing solution 183
  - documenting solution 184-185
  - finding program and language to be used 183
  - requirements 182-183
  - sample program 201-227
- DESTROY call, for MLI 155-156
- DETECT BAD BLOCKS 130-132
- Directory files 31-36
  - and CREATE command 52-54
- Disk drive 27
- Diskette 1
  - determining which operating system to use 230-231
  - DOS 3.3/ProDOS formats 23, 230-231
- DOS 3.3, comparisons to ProDOS 23, 229-240
- Drive number 17
- DW 152
- Error codes, from MLI calls 176-177
- Error messages 256-265
  - Applesoft II 264-265
  - determining the error 257
  - FILER and CONVERT 259-264
  - ProDOS 257-258
- Escape key 11
- /EXAMPLES diskette 6-8
- EXEC command 107, 110-111, 240, 255
  - transformation of Applesoft II BASIC program to text file by use of 111
- Executive files 107-111
  - demonstration 108-109

Executive files (*Contd.*)

EXEC command 110-111  
 EXERCISER program 153, 156, 157, 158, 161, 163  
 EXIT command 280-281

Field 65, 66, 69, 86  
 File 25, 30-38, 194  
   changing name of. *See* RENAME command  
   and CREATE command 52-54  
   defined 30  
   directory 31-35  
   removing from disk. *See* DELETE command  
 FILER program 8-13, 14  
 FLUSH command 65, 77-78, 81, 238, 255  
   for MLI 167-168  
   used with random-access file 81, 89, 94  
   used with sequential-access file 65, 77-78  
 FORMAT A VOLUME 125-127

General memory map 248-249  
 GET BUF call, for MLI 171, 172, 173  
 GET EOF call, for MLI 170-171  
 GET FILE INFO call, for MLI 158  
 GET MARK call, for MLI 168, 169, 170  
 GET PREFIX call, for MLI 161  
 GET TIME call, for MLI 172  
 Global variables 146

HELP command 21-22, 231, 250  
 Hexadecimal conversions 286-287  
   notation 18  
 HOLD command 277

IN# command 60-61, 236, 253  
   used with binary file 97, 103-104  
 Intelligent RUN command. *See* - (DASH)  
   command  
 I/O from programs 57-61

JSR 152

Kernel functions 4-5

LENGTH command 279  
 Linked list storage file 155  
 LIST PRODOS DIRECTORY 117-118  
 LIST VOLUMES 128-129  
 LOAD command 26, 42-43, 232, 251  
 LOCK command 55-57, 234, 252

Machine-code routines, installation 100-101  
 Machine Language Interface (MLI) 27,  
   141-178  
   direct access calls 174-176  
   error codes 176-177  
   filing calls 163-173  
   housekeeping calls 154-162  
   issuing calls 152-176  
   memory maps 143-152  
   memory usage 142-152  
   system calls 173-174  
   writing a system program 177-178  
 MAKE DIRECTORY 121-122  
 MANUAL command 276  
 Memory, in Apple II computer 27  
 Memory locations in ProDOS MLI 143-152  
 Memory map. *See* ProDOS 1.0 memory map;  
   General memory map  
 Menu, advantages 185  
 MERGE command 277-278  
 MLI. *See* Machine Language Interface  
 MONITOR 4  
 Monitor, getting into 104-105  
 MUFFIN program 23

NEWLINE call, for MLI 164-165  
 NOHELP command 22, 232, 250-251  
 NOSHOW command 279  
 NUMBER FILER subroutine 191-193  
 Numeric variable 69

ON LINE call, for MLI 160-161  
 OPEN command 236, 253  
   for MLI 163-164  
   used with random-access files 89-90  
   used with sequential-access files 64, 72-74  
 Output program 64

Partial pathname. *See* PREFIX command  
 Pathname 17, 36-38, 194  
   slot-drive summary 44  
 Pathname determination subroutines 194-196  
 PLIST 152  
 POSITION command 65, 78, 238, 255  
 PR# command 59-60, 235, 253  
   used with binary file 97, 103-104  
 PREFIX command 37-39, 47, 49-52, 233,  
   251-252  
 PRINT command 77  
 Print to screen subroutine 196  
 Processing program 64

- ProDOS 141
  - capabilities 26-27
  - commands 16-18, 39-43, 46-61, 231-240, 250-255
  - DOS 3.3 23, 229-240
  - files 25, 30-38
  - introduction to 3-13
  - programming with 177-178, 180-227 and SOS 271-273
  - starting up system with 18-21
- ProDOS diskettes
  - copying 13-15
  - EXAMPLES 6-8, 14
  - PRODOS 6-8, 14
- PRODOS/EXERCISER command 153-162
- PRODOS file 27
- ProDOS file name 30-31
- PRODOS/FILER program 112-135
  - configuration defaults 134-135
  - file commands 117-122
  - keystrokes for 113-114
  - QUIT option 135
  - volume commands 123-134
- ProDOS 1.0 memory map 241-248
- PRODOS program 28-29
- ProDOS startup diskette 28
- ProFile 113, 123, 125
  - hard disk storage 270
- Program
  - design 181-185
  - modules 185-201
  - sample 201-227
- Program modules
  - number filter subroutine 191-193
  - pathname determination subroutine 194-196
  - print to screen subroutine 196
  - read a record subroutine 196-197
  - skeleton program 185-191
  - string filter subroutine 193-194
  - system configuration setup
    - subroutine 198-199
  - Thunderclock routine 199-200
  - write a record subroutine 197-198
- /RAM 268-269
- Random-access files 64, 80-95
  - data storage in 83-88
  - reading from a record 89
  - record length 82
  - writing a record in 82-83
- READ BLOCK call, for MLI 174-175
- READ command 237, 254
  - for MLI 165-166
  - used with random-access file 81, 90-91
  - used with sequential-access file 64, 74-75
- Read current prefix routine 200
- Read a record subroutine 196-197
- Receiving data from sources other than keyboard 60-61
- RENAME command 54-55, 234, 252
  - for MLI 156
- RENAME FILES 121
- RENAME A VOLUME 129-130
- RENUMBER command 276-277
- Report heading subroutine 200-201
- RESTORE command 57-59, 235, 253
- RESTORE DEFAULTS 135
- RUN command 26, 40-42, 232, 251
- Sample program 201-227
- SAVE command 26, 43, 233
- SCRATCH. DISK 10, 11, 12, 14-15, 27, 28
- SEC 152
- SELECT DEFAULTS 134
- Sequential-access files 63-79
  - creation of 65
  - defined 65
  - and EXEC command 107, 109
  - problems 73-74
  - retrieving data from 69-72
  - storing and viewing data in 66-68
- SET BUF call, for MLI 171, 172
- SET EOF call, for MLI 170
- SET FILE INFO call, for MLI 157
- SET MARK call, for MLI 168, 169
- SET PREFIX 122
- SET PREFIX call, for MLI 161
- SHOW command 278
- Slash mark 10
- Slot number 17
- SOS 271-273
- Startup diskette 28
- STARTUP program 19, 27, 29
- STORE command 57, 58, 234, 252
- String data 191
- String filter subroutine 193-194
- String variable 69
- Subdirectory 31, 32
- System bit map 27
- SYSTEM MASTER diskette 6



System program writing 177-178

Tape drive 65

Text file 64, 65

    program to 111

    retrieving data 69

    transformation of Applesoft II BASIC program to 111

Thunderclock routine 199-200

Transfer of data from normal video screen output 59-60

Tree structure type of file 155

Ttype 52

TUTOR 113, 116, 123, 125

UNLOCK command 55-57, 234, 252

Utility programs 155

Volume 123

Volume commands 9

Volume directory 31, 32, 51  
    characteristics 31

WRITE BLOCK call, for MLI 175-176

WRITE command 64, 75, 81, 237, 254  
    used with random-access file 81, 91-92  
    used with sequential-access file 64, 75

Write a record subroutine 197-198

XREF command 279-280

Zero page 242-248

# JOHN CAMPBELL

# INSIDE APPLE'S<sup>®</sup> ProDOS<sup>™</sup>

ProDOS<sup>™</sup>, the exciting, new Professional Disk Operating System recently introduced by Apple Computer, Inc. contains new commands, expanded and improved old commands, file management utilities, assembler, data types, file types, and new procedures. It is truly a fascinating environment that adds materially to the capability of the Apple<sup>®</sup> II family of computers.

INSIDE APPLE'S ProDOS gives you a practical, usable explanation of this interesting computer disk operating environment. Using a logical approach, the author first introduces the ProDOS operating system and outlines the conventions used throughout the book for describing ProDOS. In subsequent chapters, you'll find impressive coverage of files, file types, and command syntax. In addition, the author discusses the most-used ProDOS commands and general system house-keeping commands. The last chapter brings it all together by presenting a number of routines that you can use in your original programs.

With INSIDE APPLE'S ProDOS, you'll make the transition from DOS 3.3 to ProDOS with ease.

*Apple<sup>®</sup> is a registered trademark and ProDOS<sup>™</sup> is a trademark of Apple Computer, Inc.*

*Cover Design by Debbie Balboni*

**A RESTON COMPUTER GROUP BOOK**  
**RESTON PUBLISHING COMPANY, INC.**  
*A Prentice-Hall Company*  
Reston, Virginia



**0-8359-3078-5**